FPLC 設計操作手冊

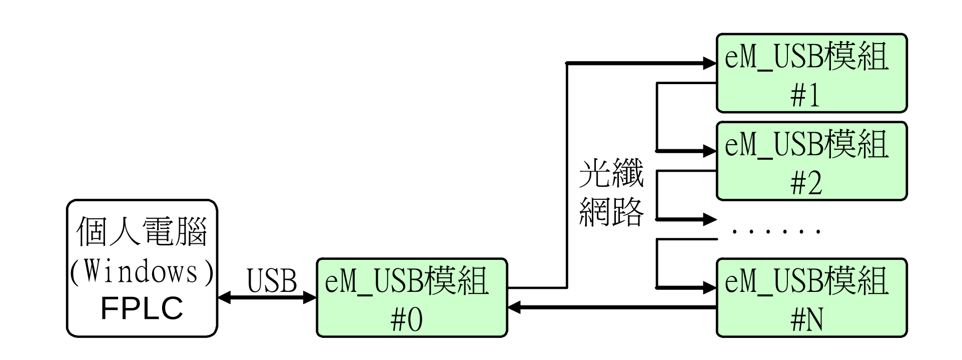
俊原科技技術手冊

Version 1.0

目錄

1.FPLC 系統規劃

FPLC 是一套相當親和力人機圖控軟體，它可以規劃順序控制，伺服控制，多軸控制，CNC 軌跡控制與影像量測監控等多功能操控系統。使用者非常容易自行規劃操控畫面，搭配在個人電腦 Windows 98、ME、2000、XP 作業系統下透過 USB 界面結合光纖網路通訊 FPGA 發展系統，系統連線架構規劃如下：



其中：

1、基本模組為 eM_USB 模組，可用 USB 界面和個人電腦連線，並提供 USB 對光纖網路的轉換功能，使得個人電腦可以獨立控制光纖網路中的每個網路節點。

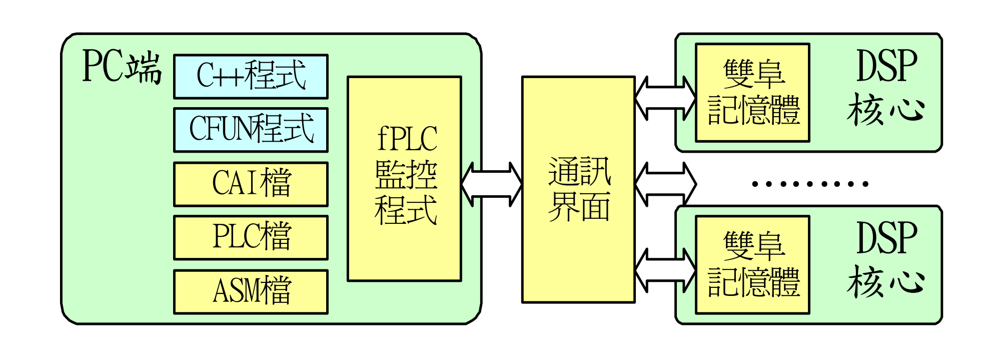
2、eM_USB 模組也可以作為光纖網路中的網路節點，透過光纖網路而接受個人電腦的監控。

3、就個人電腦而言，不論是 USB 界面直接連結或是以光纖網路間接連結的 eM_USB 模組，都是可以直接監控的網路節點。最多可以同時控制一個 USB 節點和 30 個網路節點。

4、對於每一個網路節點，個人電腦都可執行：
eM_USB 模組中 FPGA的 電路下載工作，並在適當的 FPGA 電路配合下，可執行 eM_USB 模組中 FPGA 內部電路的讀寫控制。

5、透過適當的 FPGA 電路設計，各網路節點都可透過光纖網路和其他任何網路節點作同步通訊，執行 FPGA 內部記憶體的資料交換。資料交換頻率可達 1KHz 以上。

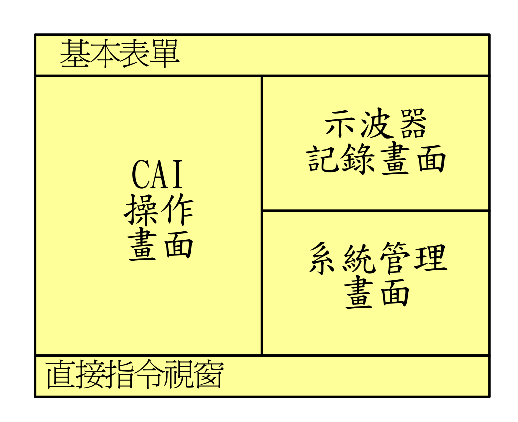
fPLC 操作系統是 DSP 核心的系統監控程式，在 PC 端的視窗環境中執行。其系統方塊圖如下：



其中：

1. fPLC 透過通訊界面執行 DSP 核心的監控工作。在光纖網路的環境下，可以監控最多 31 組的 DSP 核心。而 fPLC 可以任意選擇一組的 DSP 核心執行監控工作。
2. 對於 DSP 核心的監控工作，實際上就是讀寫 DSP 核心中的雙阜記憶體。雙阜記憶體是 DSP 核心中的主要記憶體，儲存所有的資料、程式、控制命令和狀態。PC 端只要改變雙阜記憶體的內容，就可改變 DSP 核心的控制動作。
3. fPLC 執行監控工作時，所需要的參考檔案包括：
   CAI 檔：負責 fPLC 中監控畫面的定義。
   PLC 檔：負責 fPLC 中批次指令的執行。
   ASM 檔：負責 DSP 核心中的組合語言程式。
4. 另外也依需要可增加：
   C++ 程式 ：可自行設計完全彈性的操作和監控程式。
   CFUN 程式：可自行設計控制程序和計算程式。

fPLC畫面規劃



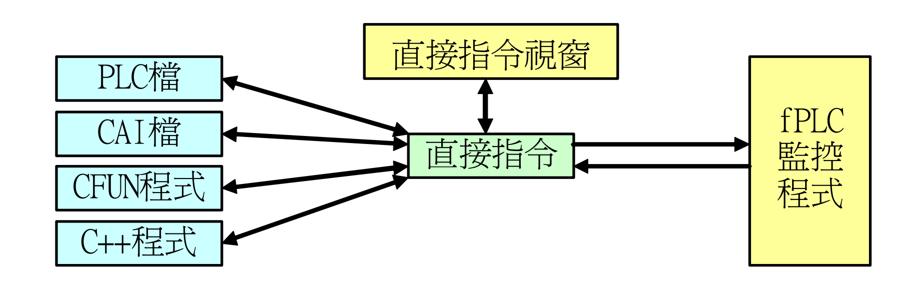
從外觀上來看，fPLC的畫面規劃如下：

其中：

1. 基本表單中標示著常用的 fPLC 指令，可直接選取後按鍵執行。
2. CAI 操作畫面執行整個監控和操作程序，由 CAI 檔定義。
3. 示波器記錄畫面，先記錄即時控制的結果，再以示波器格式顯示。
4. 系統管理畫面顯示

   目前的網路狀況，各個 DSP 核心(包括 FPGA)的電路下載，
   DSP 核心中的變數分配和使用狀況。
5. 直接指令視窗可以下達直接指令。

直接指令規劃



fPLC 的整個操作環境都是由直接指令來控制，圖示如下：
其中：

1. 所有 fPLC 中的監控動作，都是透過直接指令來執行。
2. 所謂直接指令，就是一連串的文字串命令，送至 fPLC 程式中後就立刻執行該項指令，並將執行結果以文字串結構送回。
3. 任何程式只要能產生文字串，就能透過直接指令來操作 fPLC 程式。包括：

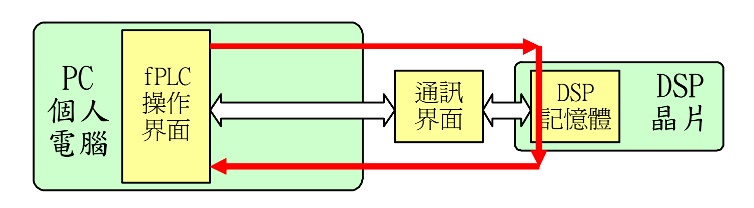
直接指令視窗：以按鍵方式，一步步的執行直接指令。
PLC 檔：將所需要的直接指令彙整成文字檔後，以批次指令一併執行。
CAI 檔：根據操作畫面定義，在不同操作選擇下執行指定的直接指令。
CFUN 程式：其他的 C 程式可透過公用記憶體來設定並執行直接指令。
C++ 程式：其他的 C++ 程式可透過公用記憶體來設定並執行直接指令。

下面列舉一些常用的直接指令：

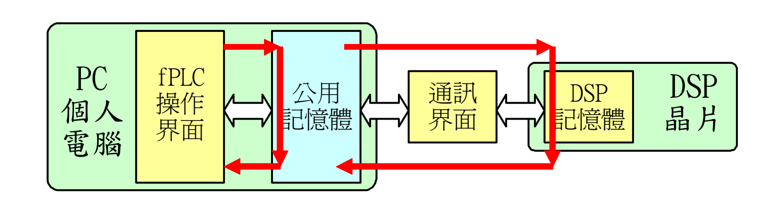
```
D <addr> <len>;              //**以  Dump指令來顯示資料的內容
S <addr> <data>;             //**以   Set指令來設定資料
F <addr> <len> <data>;       //**以  Fill指令來充填資料區
C <addr> <len> <data>;       //**以Compare指令來檢查資料區
TABLE <addr> <data>;         //**以 Table指令來設定資料區
Rxxxx=Ryyyy;                 //**以計算式來設定單筆的資料
DATA Rxxxx;                  //**在CAI畫面中作變數的顯示和設定
```

公用記憶體規劃

DSP 核心的通訊必須透過非同步的 USB 界面和光纖網路作連線，使得 DSP 核心中的資料無法立即送至 fPLC 操作界面。一般而言，這些通訊上的延遲現象可能高達數十毫秒，圖示如下：



為了方便操作系統的設計，我們規劃了一組公用記憶體作為中間界面，以方便網路資料的讀寫工作。圖示如下：



其中：

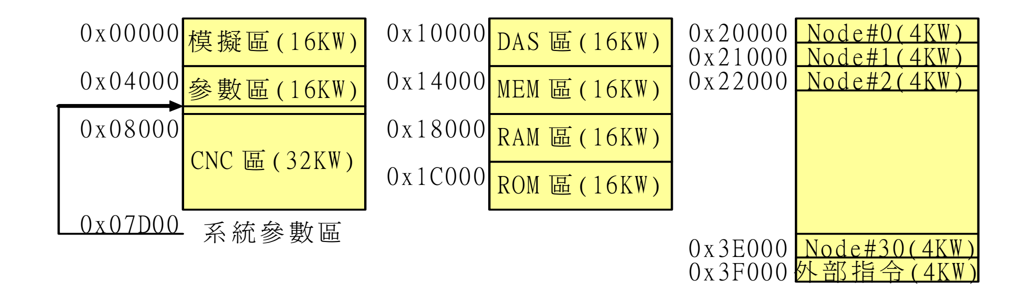
1. 公用記憶體的內容直接對應到 DSP 核心中的記憶體內容。
2. 當 fPLC 讀寫 DSP 核心時，只是讀寫公用記憶體中的內容，而不是 DSP 核心。
3. 一旦 fPLC 讀寫公用記憶體後，通訊界面即會自動的執行 DSP 核心的讀寫動作。所謂讀取即是將 DSP 核心中的資料讀至公用記憶體中，而所謂寫入即是將公用記憶體中的資料寫入 DSP 核心中。
4. 因此一個完整的寫入程序如下：
   fPLC 寫入公用記憶體(就 fPLC 而言，這個寫入程序已經立刻結束)
   ==>公用記憶體再透過通訊界面寫入 DSP 核心(可能有些通訊延遲)。
   但是基本上，DSP 核心仍可快速取得 fPLC 的寫入資料。
5. 而一個完整的讀取程序如下：
   fPLC 讀取公用記憶體(就 fPLC 而言，這個讀取程序已經立刻結束)
   ==>公用記憶體再透過通訊界面讀取 DSP 核心(可能有些通訊延遲)。
   所以 fPLC 所讀取的並不是 DSP 中的最新資料，而是前一次讀取的舊資料。

加入公用記憶體之後，整個操作特性變更為：

1. fPLC 要讀寫 DSP 核心時，不需等待通訊延遲，即可立刻完成讀寫動作。
2. 當寫入時，DSP 核心仍能取得 fPLC 的最新寫入資料。但是當讀取時，fPLC 端所取得的只是 DSP 核心的上一筆舊資料。
3. 當 DSP 核心是在執行即時控制程式時，所謂新舊資料的差異非常有限。只要 fPLC 持續的讀取同一位置，該筆資料就可以隨時取得最接近的更新值。

公用記憶體配置

公用記憶體的空間配置如下：



其中：

1.PLC_data[]佔用 256K-word 的空間，前面 64K-word 位址配置如下：
- 0x00000~0x01FFF(8KW) ：對應到 DSP 模擬器中的 8K-word 資料區
- 0x02000~0x03FFF(8KW) ：對應到 DSP 模擬器中的 8K-word 程式區
- 0x04000~0x07CFF(16KW)：可讓操作者到自由運用的參數區
- 0x07D00~0x07FFF(768W)：對應到系統參數區
- 0x08000~0x0FFFF(32KW)：對應到 CNC 控制用的參數區，目前尚未使用。

2.其他部分的空間配置如下：
- 0x10000~0x13FFF(16KW)：對應到示波器的 DAS 頻道和頁面控制。
- 0x14000~0x17FFF(16KW)：對應到 DSP 中記憶體的上下載處理區
- 0x18000~0x1BFFF(16KW)：對應到 DSP 中外部 RAM 的上下載處理區
- 0x1C000~0x1FFFF(16KW)：對應到 DSP 中外部 ROM 的上下載處理區
- 0x20000~0x3EFFF(4KW*31)：對應到 31 組 DSP 的 4KW 變數區
- 0x3F000~0x3FFFF(4KW) ：對應到外部 C 程式的指令控制區

就操作者而言，可獨立運用的記憶體空間包括：

分類	資料	使 用 說 明
PLC區	PLC_data[256KW]	所有的公用記憶體空間，但是只有前面 64KW 才用 PLC_data 來直接讀寫
DAS區	DAS_data[16KW]	示波器控制用的資料蒐集區
MEM區	MEM_data[16KW]	DSP中記憶體的上下載處理空間
RAM區	RAM_data[16KW]	DSP中外部 RAM 的上下載處理空間
ROM區	ROM_data[16KW]	DSP中外部 ROM 的上下載處理空間
NET區	NET_data[31][4KW]	31組 DSP 變數區的對應空間
CMD區	PLC_command[4KB]	外部 C 程式的指令下達區
STS區	PLC_echo[4KB]	外部 C 程式的指令執行結果

外部指令的執行

外部指令區共佔用公用記憶體中 4K-word 或是 8K-byte 的空間，定義如下：

0x3F000	PLC_command (4KB)
0x3F800	PLC_echo (4KB)

其中：

1.外部指令是類似文書檔的架構，所以是以 byte 爲單位。

2.0x3F000~0x3FFFF 共佔用 4K-word，即是 8K-byte 的空間。其中分成兩部分

```
char PLC_command[4096]：儲存外部命令。
char PLC_echo[4096]   ：儲存外部命令的執行結果。
```

3.外部指令的執行程序如下：

```
*PLC_echo=0;                  //清除執行結果
strcpy(PLC_command,"...");    //設定外部指令
while (*PLC_command) ;        //等待指令執行的結束
strcpy(...,PLC_echo);         //檢查執行結果
```

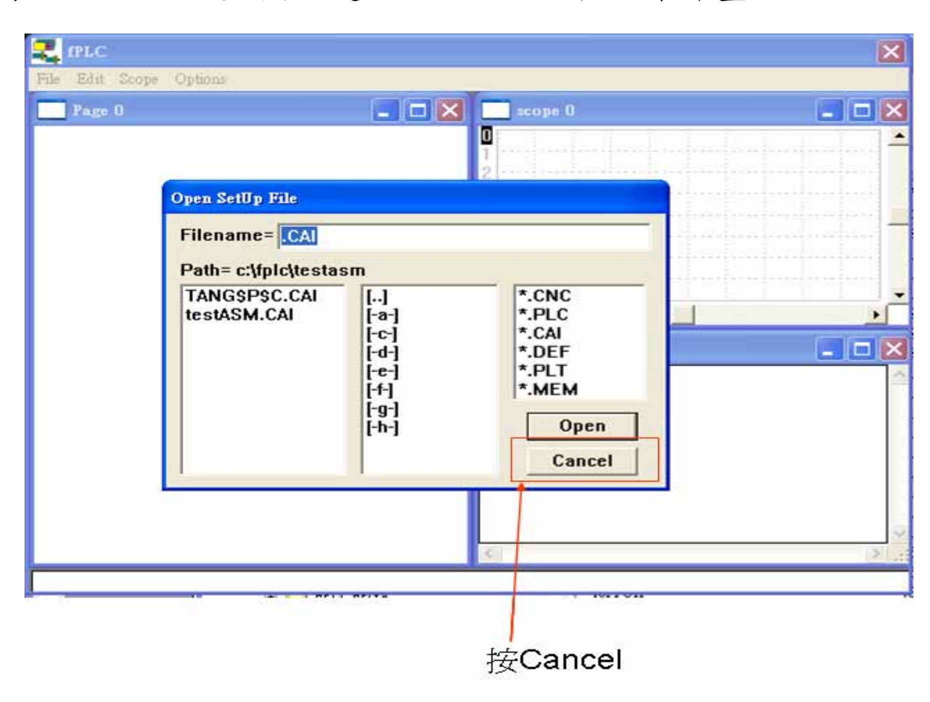
4.當外部指令執行完畢時，一方面會設定 PLC_echo 值，另一方面也會清除 PLC_command 值。因此可用 PLC_command 是否被清除來檢驗指令的結束與否。

5.PLC_command[4096]中儲存批次指令，例如

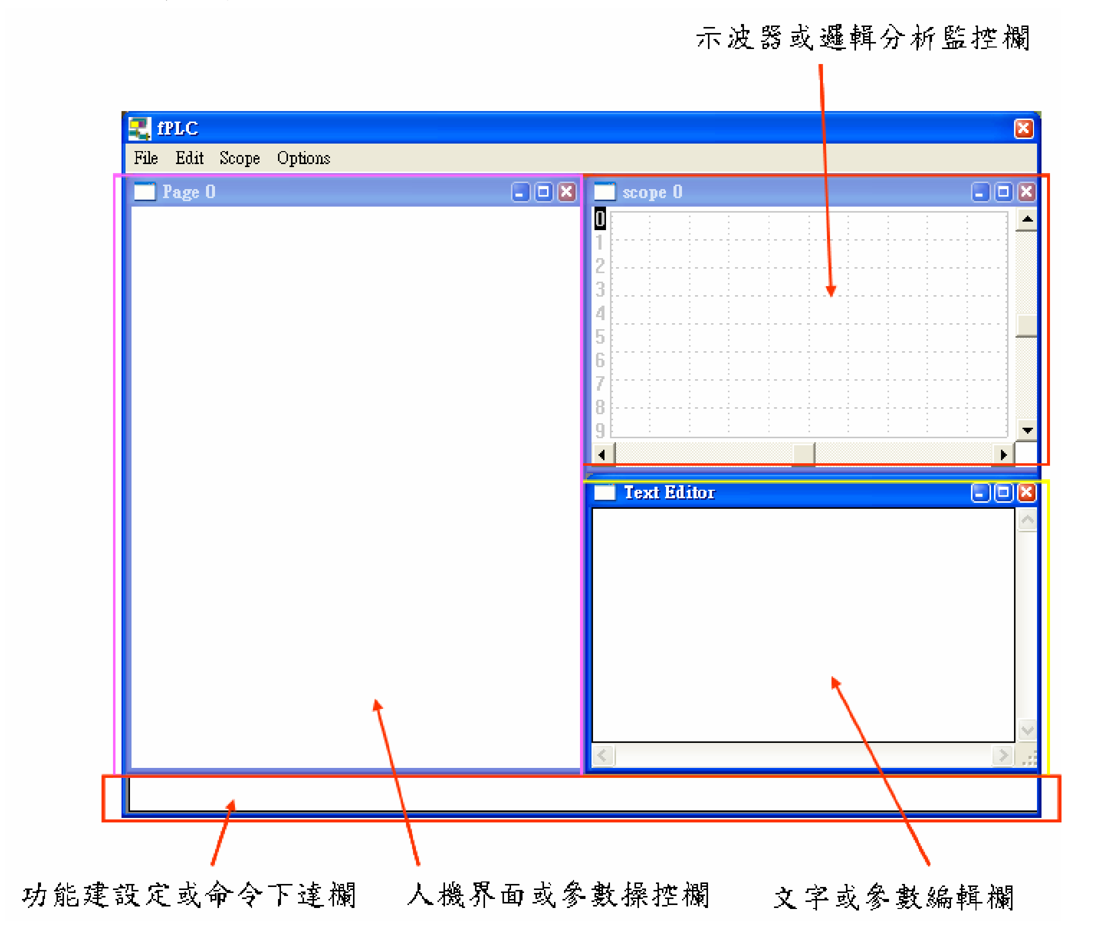
```
S 0x200 0x11 0x22;            //所有直接指令都可使用
D 0x200 2;                    //指令的執行結果會在 PLC_echo 中儲存
...;                          //各指令以 ';' 作區隔，可連續下達指令
...;                          //只要指令總長不超過 4096-byte 即可
```

所有其他的 C 程式或 C++ 程式都是透過外部指令區的操作，下達直接指令以操控 fPLC 程式的動作。透過 fPLC 程式即可控制每一個 DSP 核心的硬體動作。

首先將光碟片 RFFPGA 目錄檔案拷貝在您工作硬碟裡面，首先進入RFFPGA 裡子目錄 testASM，直接點選 fPLC.exe 出現下面操作畫面:

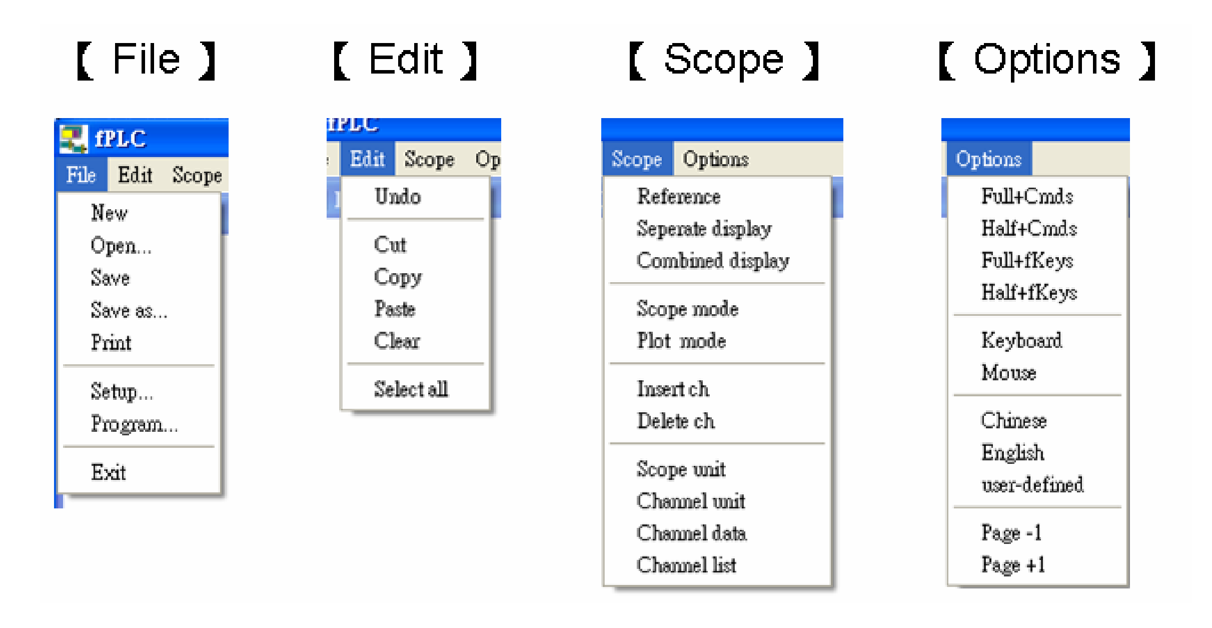


出現 FPLC 操控畫面



1.1 FPLC 系統指令說明

以下就將 FPLC 相關指令說明如下：



File 子目錄下指令

New ：建立一個新的檔案，此時是建立 FPLC 文字或參數編輯欄，通常較少使用此功能。

Open：打開該目錄下檔案，此時是打開 FPLC 文字或參數編輯欄，通常較少使用此功能。

Save：儲存編輯之檔案，此時是儲存在 FPLC 文字或參數編輯欄，通常較少使用此功能。

Save as...：儲存編輯之另外檔名，此時是儲存在 FPLC 文字或參數編輯欄，通常較少使用此功能。

Print：列印編輯之檔案，此時是列印在 FPLC 文字或參數編輯欄，通常較少使用此功能。一般如您列印 FPLC 畫面，可以使用 Windows 鍵盤 ALT ＋ Print Screen ，此時該 FPLC 畫面已被複製，您只要在任何執行軟體使用貼上，就可將該 FPLC 畫面顯示出來，再由該軟體下執行列印功能。

Setup…：呼叫所有 *.CAI 檔並執行該檔案。

Program…：呼叫所有 *.PLC 檔並執行該檔案。

Exit：離開 FPLC 文字或參數編輯欄，通常較少使用此功能，通常在 FPLC 右上角 以滑鼠單擊 x 就可以離開 FPLC 系統。

Edit 子目錄下指令

Undo ：在 FPLC 文字或參數編輯欄編輯檔案時還原功能，通常較少使用此功能。

Cut ：在 FPLC 文字或參數編輯欄編輯檔案時剪掉功能，通常較少使用此功能。

Copy ：在 FPLC 文字或參數編輯欄編輯檔案時拷貝功能，通常較少使用此功能。

Paste：在 FPLC 文字或參數編輯欄編輯檔案時貼上功能，通常較少使用此功能。

Clear：在 FPLC 文字或參數編輯欄編輯檔案時清除功能，通常較少使用此功能。

Select all：在 FPLC 文字或參數編輯欄編輯檔案時選擇所有功能，通常較少使用此功能。

Scope 子目錄下指令

Reference：參考波形，此功能主要測試「示波器視窗」是否正常運作而設，當您按下此功能表單後可自動的在視窗上繪出四組不同之測試波形，除此之外沒有實際用途。

Seperate：分項顯示，當示波器上有兩組以上資料要顯示時，為避免多組資料繪圖時重疊在一起，不易觀測，可應用此功能使其各頻道分開顯示以利觀察。

Combined display：合併顯示，此功能正好與「分項顯示」功能相反，當示波器上有兩組以上相關資料要重疊以比較其差異性時便可啟用此功能使其「零位點」位於同一座標軸而重疊顯示。但各軸之縱軸尺度單位（scale）則由系統自行依據資料收據來調整最適宜的狀態。

Scope mode：示波器顯示，此功能配合下面「示波器取消」功能而使用，當使用示波器取消時，或執行 CNC 軌跡監控與影像監視時，欲切換至示波器顯示時使用此功能。

Plot mode：示波器取消，系統會切至不同操作模式，如執行 CNC 軌跡監控與影像監視時，此時可能不希望示波器視窗為原來之示波器用途。

Insert ch：頻道顯示（如同鍵盤 Insert 鍵），此功能配合下面「刪除顯示頻道」功能而使用，將點選之頻道關閉後，點選此功能，示波器視窗會依次將刪除頻道依依顯示出來。

Delete ch：刪除顯示頻道（如同鍵盤 Delete 鍵），此功能乃避免示波器上顯示太多資料而不亦觀察，暫時將點選之頻道關閉，使其從示波器視窗上消失，但不會影響其預先設定記錄功能，同時也配合「頻道顯示」即可呼叫被刪除之頻道。

Scope unit：顯示單位，按此功能 FPLC 會將目前顯示示波器頻道相關資料顯示於文字或參數編輯欄。

Channel unit：頻道單位，將點選之頻道相關資料顯示於文字或參數編輯欄。

Channel data：頻道資料，將點選之頻道相關資料顯示於文字或參數編輯欄，目前無此功能。

Channel list：頻道資料列印，將點選之頻道相關資料顯示於文字或參數編輯欄，目前無此功能。

Options 子目錄下指令

Full+Cmds：當 FPLC 只顯示「人機界面或參數操控欄」與功能鍵時，按此功能則恢復 FPLC 原先畫面。

Half+Cmds：在 FPLC 只顯示「人機界面或參數操控欄」與功能鍵，其他視窗功能不顯示。

Full+fkeys：點顯此功能，FPLC 會將命令下達欄變成功能鍵顯示欄，當 FPLC 只顯示「人機界面或參數操控欄」與功能鍵時，按此功能則恢復 FPLC 原先畫面。

Half+fkeys：在 FPLC 只顯示「人機界面或參數操控欄」與功能鍵，其他視窗功能不顯示。

Keyboard：切換至鍵盤操作功能設定鍵，目前不需選擇就同時具備此功能，也就是可以使用鍵盤 F1 至 F12 功能鍵。

Mouse：切換至滑鼠操作功能設定鍵，目前不需選擇就同時具備此功能，也就是可以使用滑鼠點選 F1 至 F12 功能鍵。

Chinese：FPLC 指令功能以中文顯示，目前無此功能。

English：FPLC 指令功能以英文顯示，目前全部以英文顯示。

User-defined：使用者自行定義功能，此功能是考慮將來擴充特殊功能定義用。

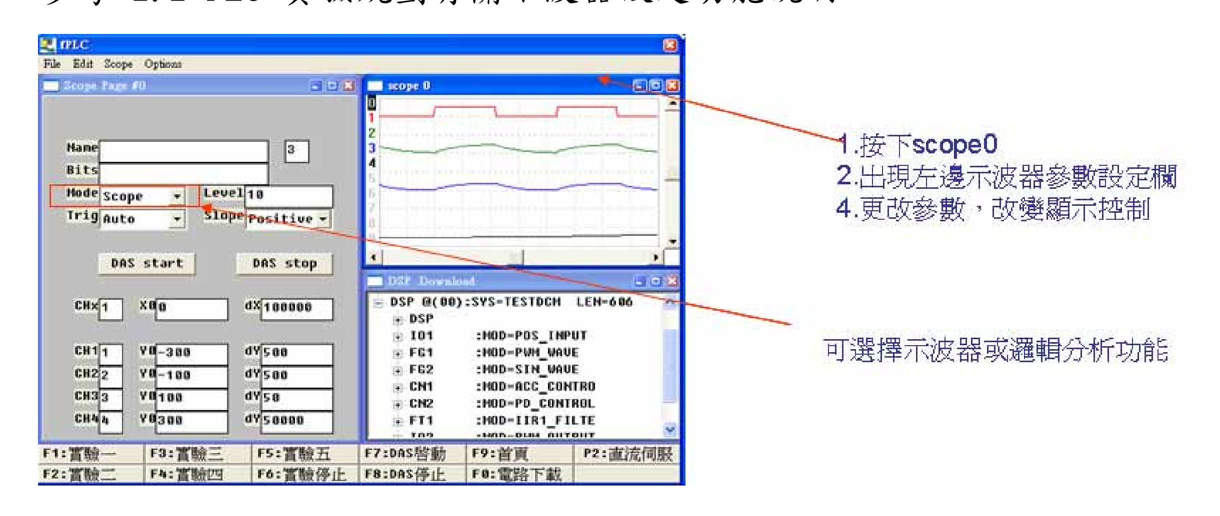
Page -1：頁數減一，配合 Caidraw 頁數設計選擇用。

Page +1：頁數加一，配合 Caidraw 頁數設計選擇用。

1.2 FPLC 示波器說明

示波器或邏輯分析監控欄說明

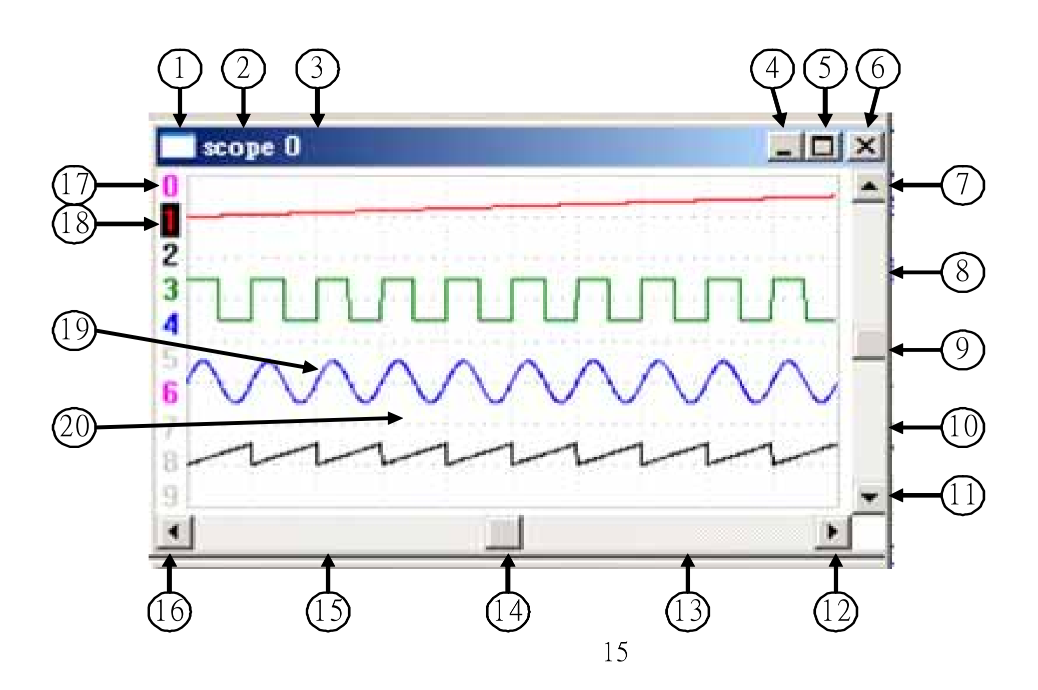
當您點擊 scope 0 文字所在位置，就會出現下面的畫面，左邊出現的示波器參數設定欄，可以修改參數，改變顯示控制，有關於如何設定該功能請參考 2.2 PLC 實驗規劃有關示波器設定功能說明。



另外，我們來看一下示波器畫面的定義：

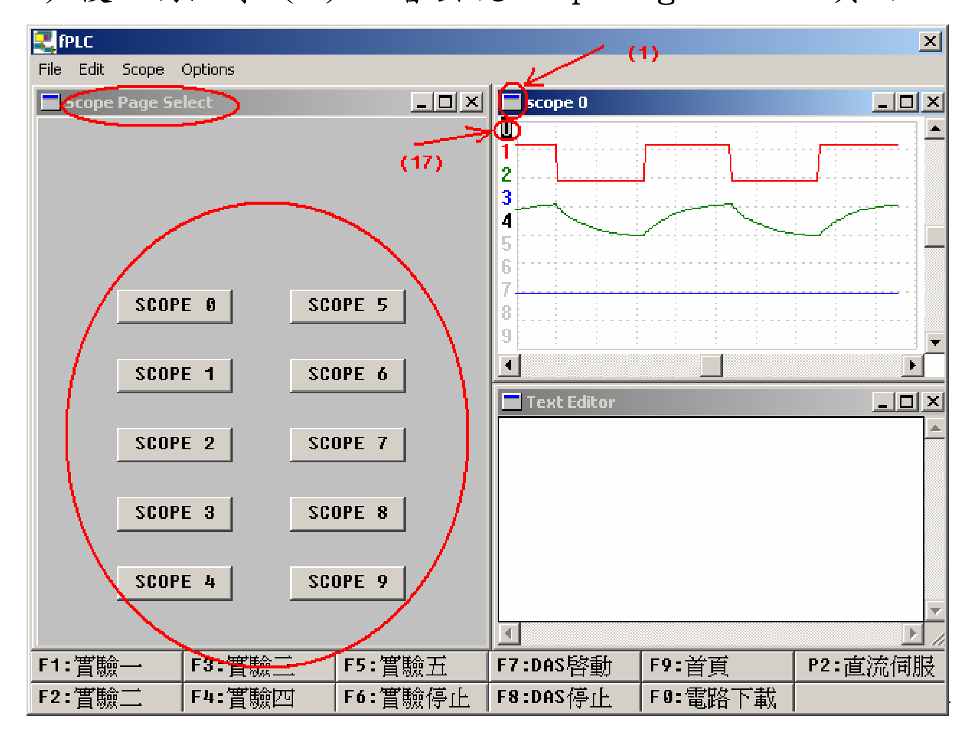
DSP 模式以四頻道的示波器顯示 DSP 核心的控制狀況，特性如下：

1. 必須和 DSP 核心配合，在 DSP 程式中一定要裝設 DAS4.MOD 的記錄模組，利用 DSP 核心來執行資料記錄工作。
2. 由 DSP 核心以 @timer 頻率做抽樣，可選擇 1KHz~32KHz。
3. 通常用於 DSP 核心的軟體偵錯用。

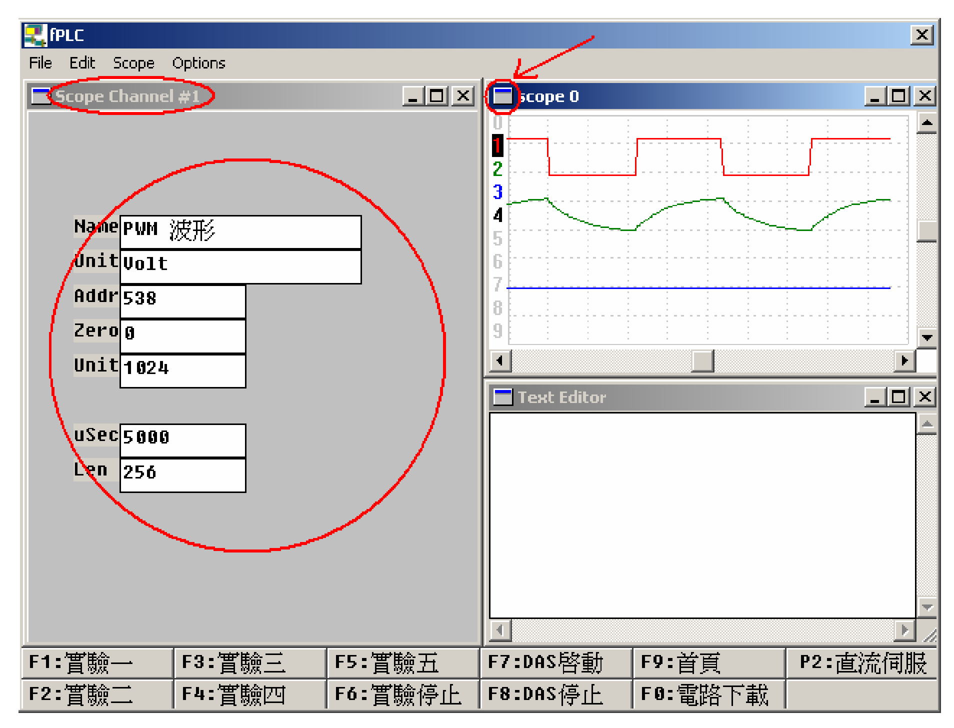


上圖中，點擊不同的點，會產生不同的作用：

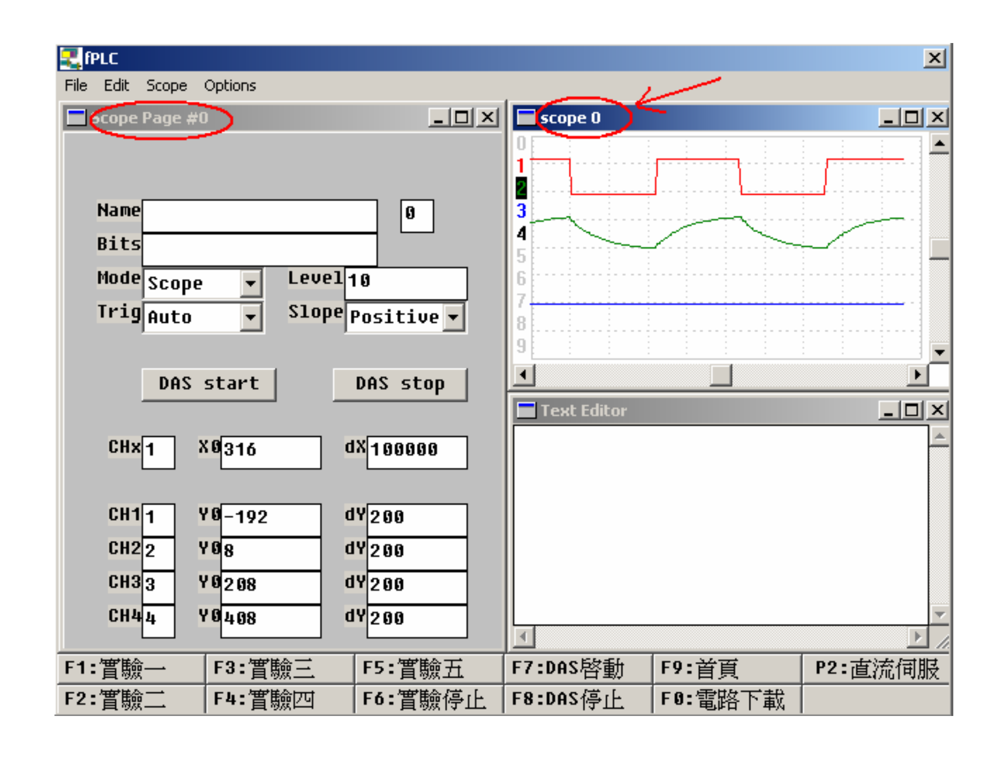
- 點擊（17）後，再點擊（1），會出現 Scope Page Select 頁面，如圖：



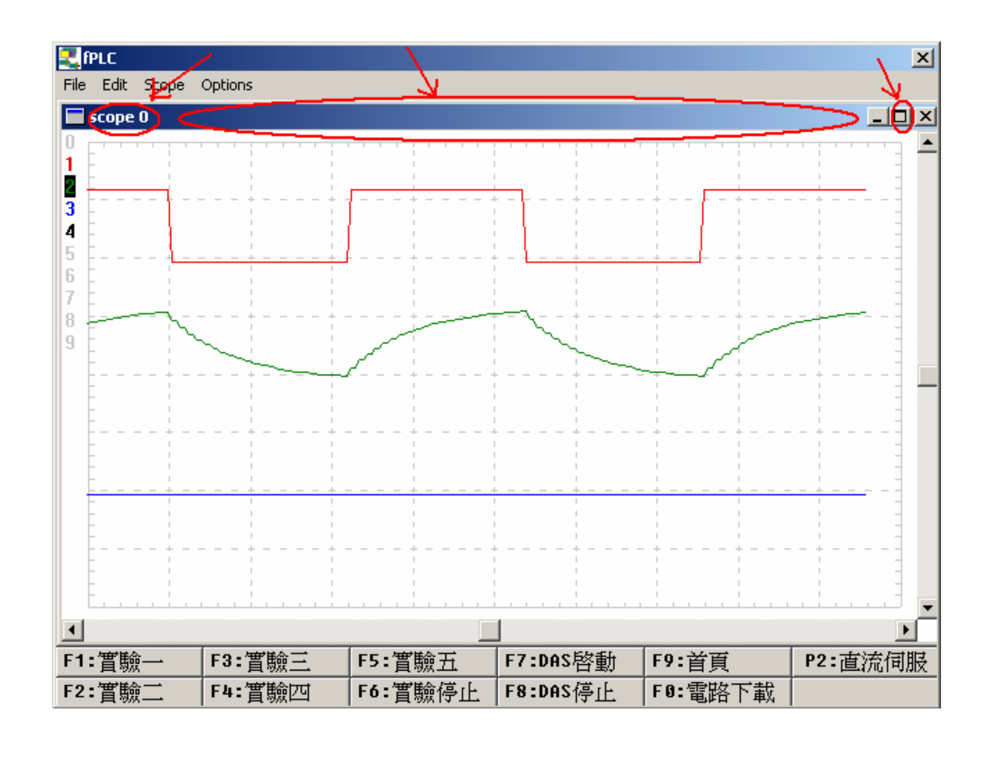
- 若點擊（18），即界面中除了零以外的數字，然後點擊（1），則會出現下圖所示的 Scope Channe＃1 參數頁面，可以手動更改其參數設定。



- 若點擊（2）scope 0 的文字，則會出現 Scope Page＃0 參數頁面，如下，同樣可以手動更改參數設定。



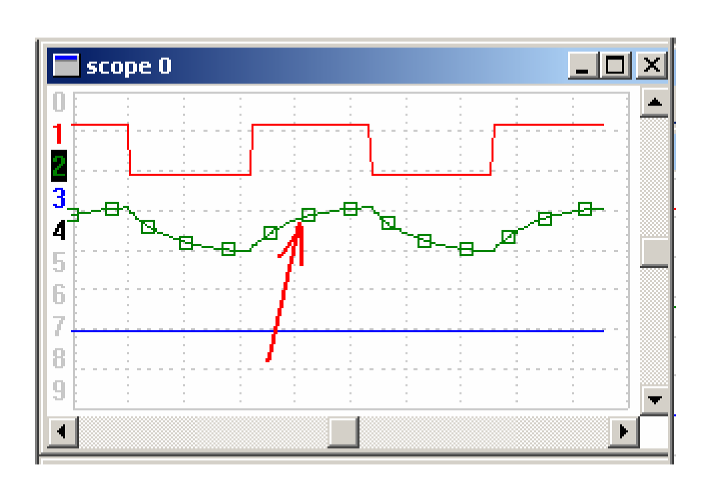
- 如果雙擊（2）scope 0 文字或者（3）旁邊的藍色區域，或者單擊（5）最大化（Maximize）按鍵，作用都是相同的，即將示波器顯示界面放大，如下圖：



若要恢復到原來的界面，只需再次進行相同的操作，即雙擊（2）scope

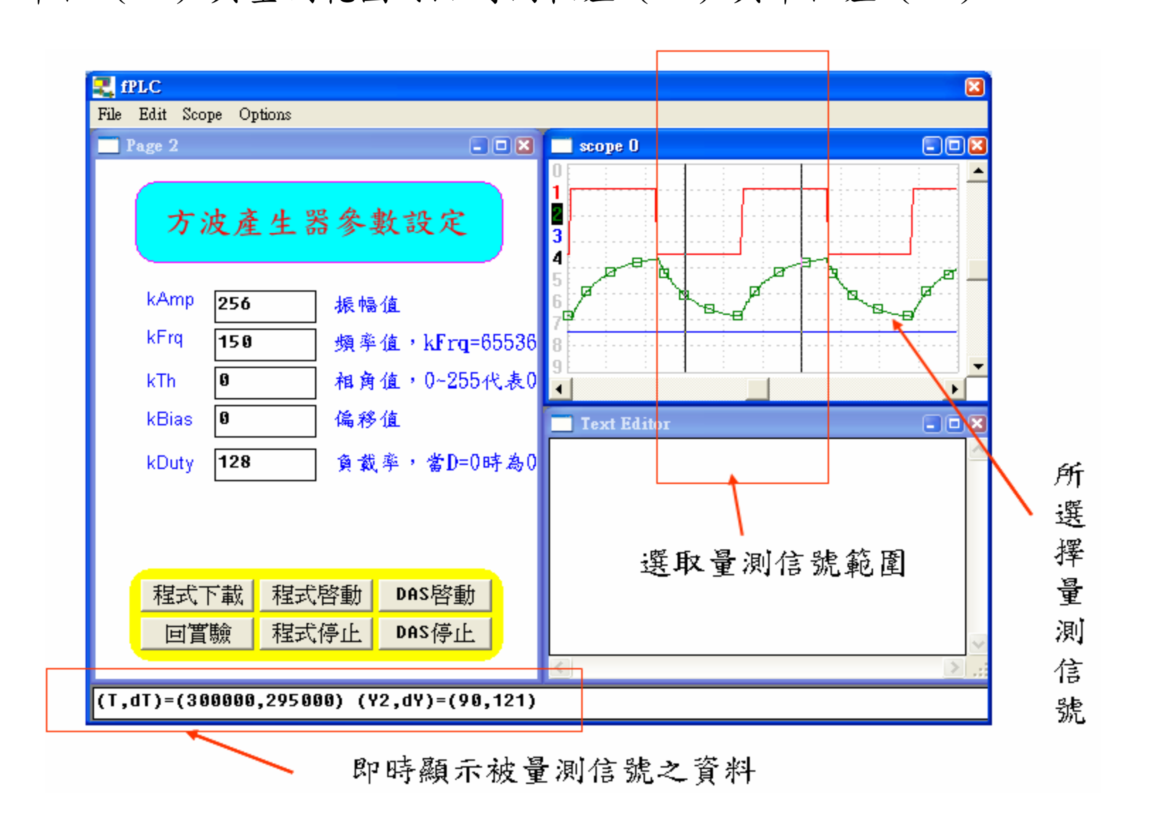
0文字或者（3）旁邊的藍色區域，或者單擊（5）最大化（Maximize）按鍵。

- 如果點擊（19），即示波器中的圖像，則被點擊的圖像會有如下顯示：

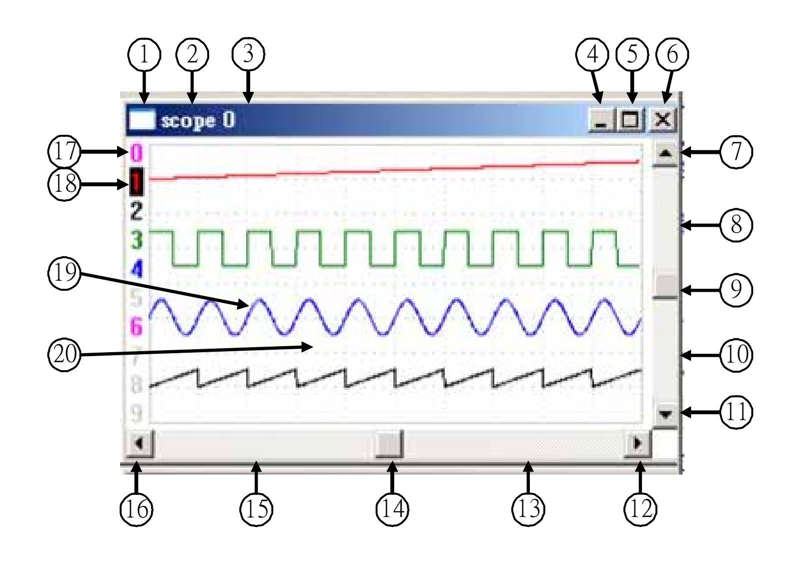


其中，綠色的圖像是被點擊的。

- 如果用鼠標先點選預備量測信號，再以滑鼠左鍵拖洩預量測信號範圍則會看到兩條豎綫出現在示波器內，可用於測量讀數，其測量讀數相關資料會顯示於功能建設定或命令下達欄：(T, dT)=(300000, 295000) (Y2, dY)=(90, 121)，該資料說明第 2 頻道量測範圍起始點時間軸（T）、單位（Y2）與量測範圍終點時間軸差（dT）與單位差（dY）。

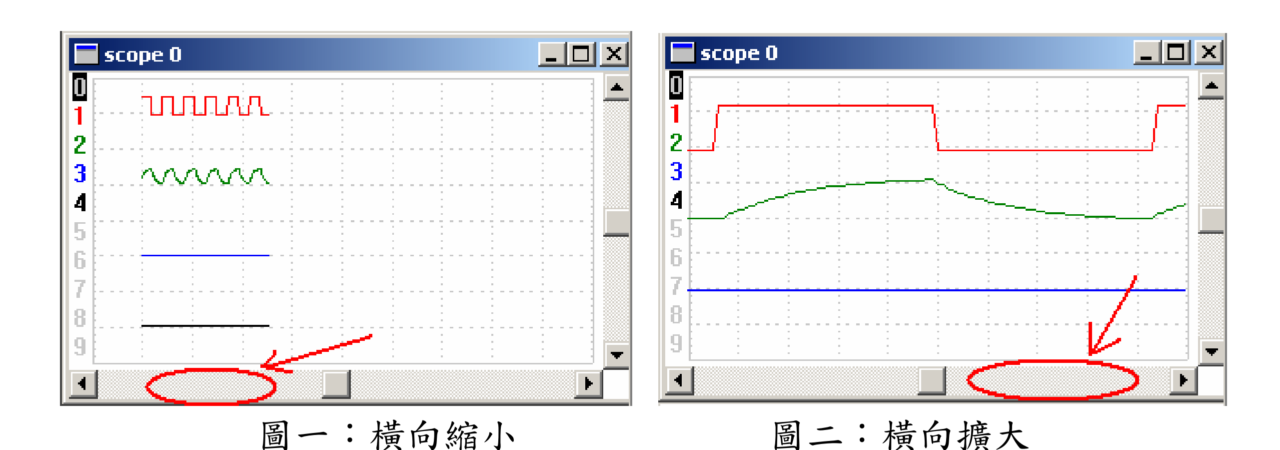


- 如果點擊（16），示波器顯示圖像向左移動；點擊（12）示波器圖像

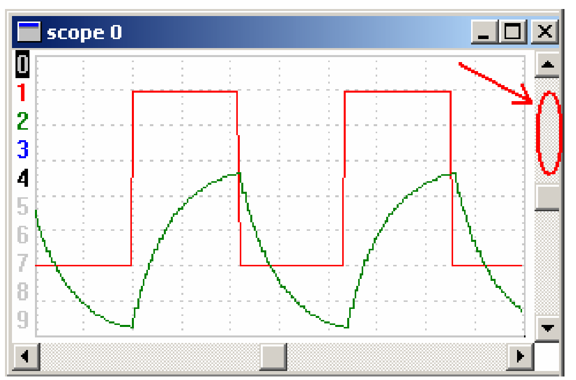


向右移動；點擊（7）示波器圖像向上移動；點擊（11）示波器圖像向下移動。 左右上下移動，也可以通過向左右拖動（14）和上下拖動（9）來完成。

- 如果點擊（15），示波器圖像橫向縮小（見下圖一）；點擊（13），示波器圖像橫向擴大（見下圖二）；點擊（8），示波器圖像縱向擴大（見下圖三）；點擊（10），示波器圖像縱向縮小（見下圖四）。



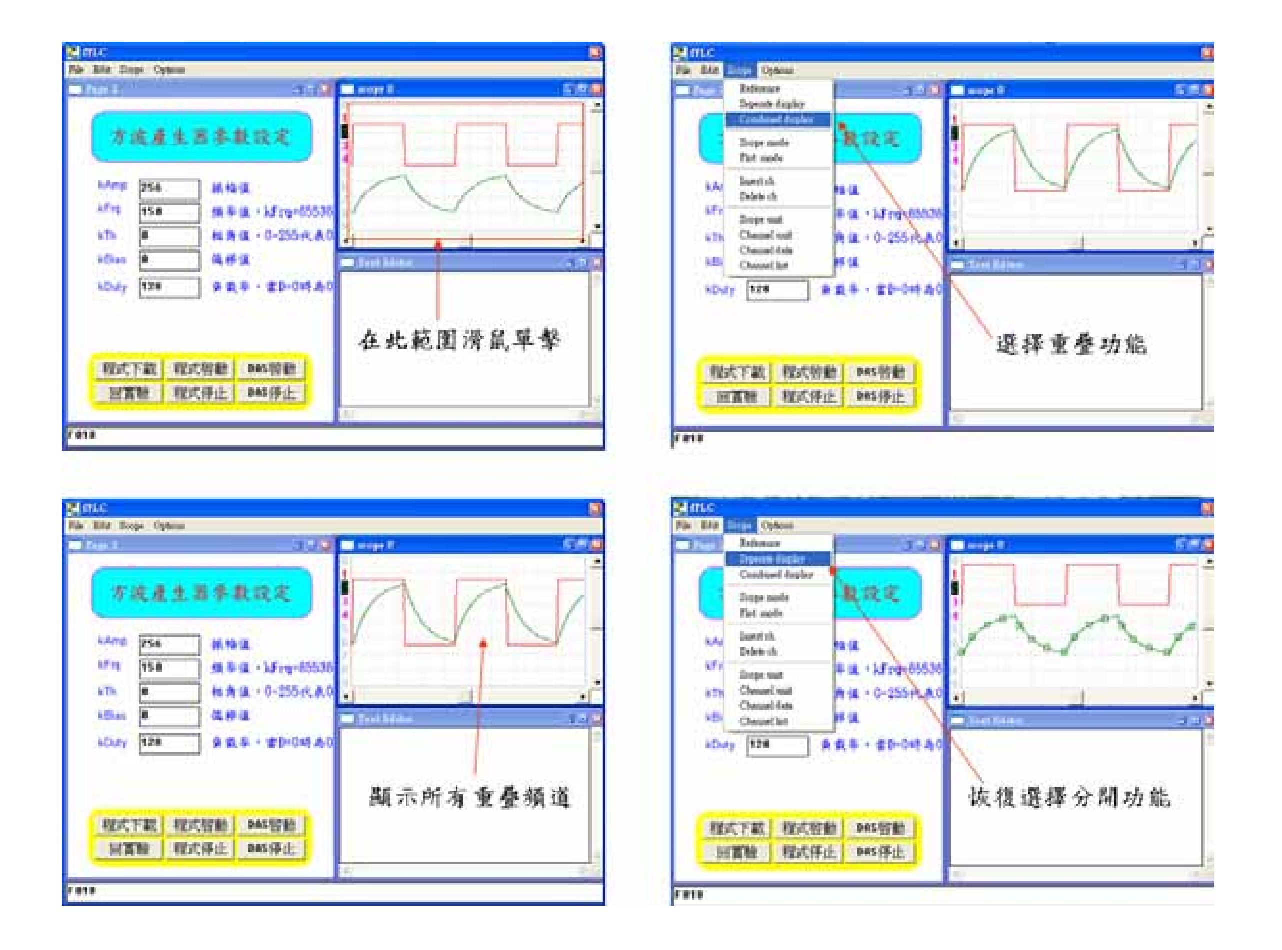
圖一：橫向縮小　　　　圖二：橫向擴大



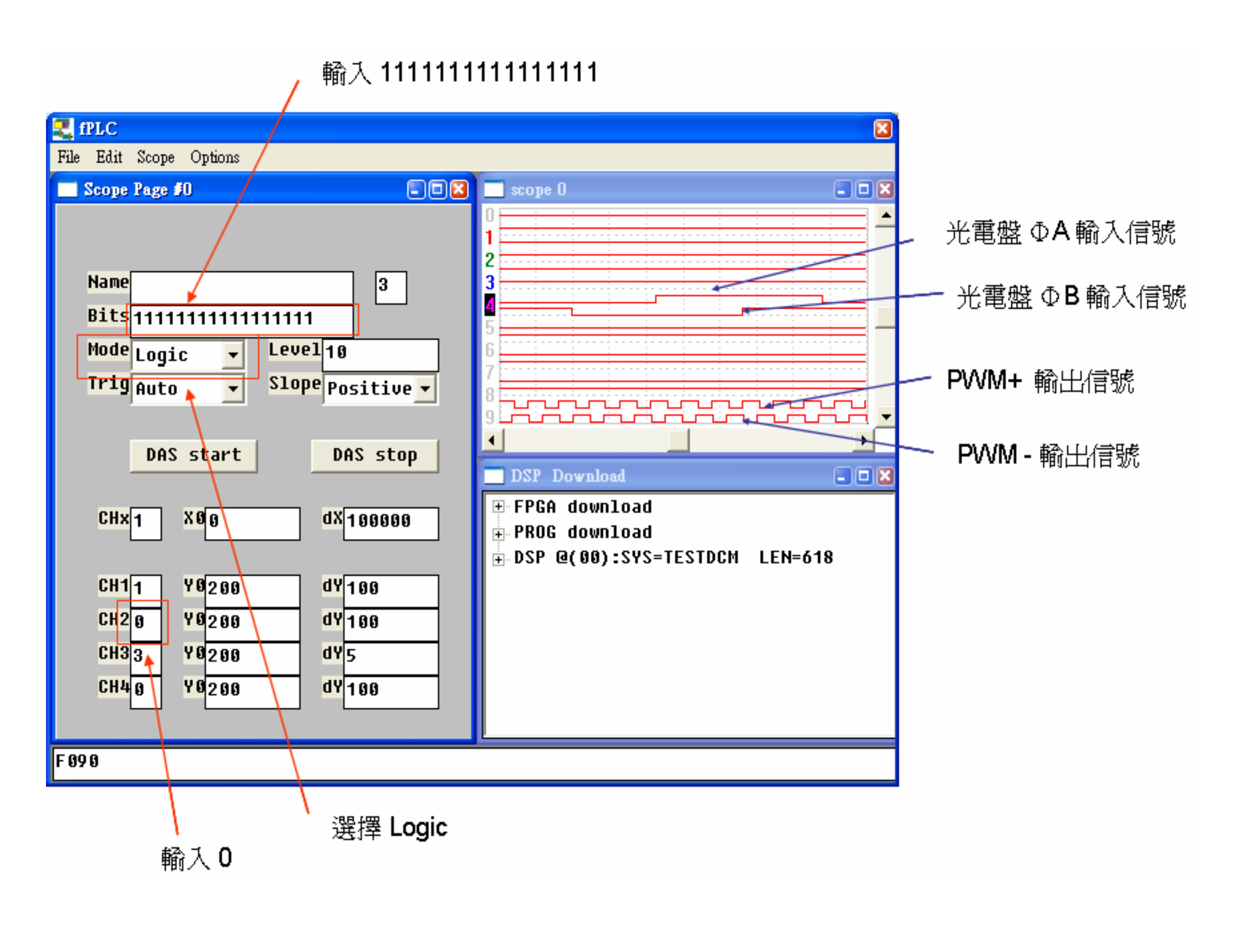


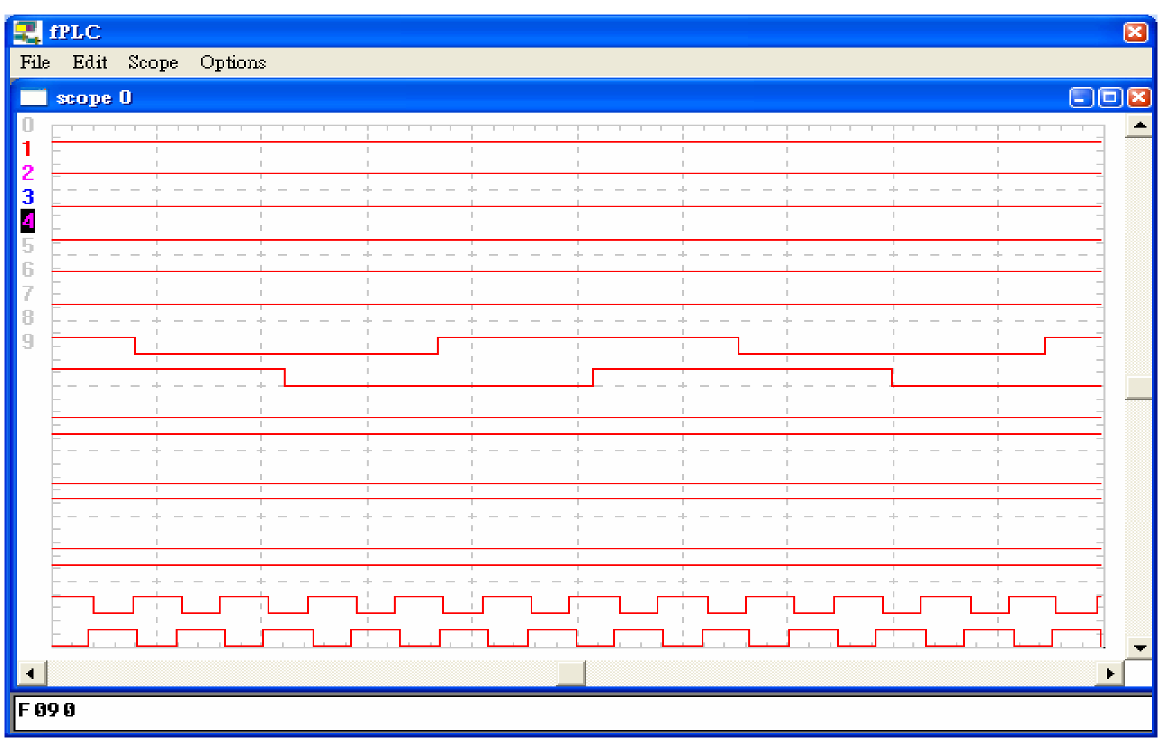
圖三：縱向擴大　　　　圖四：縱向縮小

- 記錄頻道重疊與分開設定：先點擊（20）範圍，再至 FPLC 功能命令欄選擇 Scope 下之 Combined display 則記錄頻道會重疊在一起顯示，如果再至 FPLC 功能命令欄選擇 Scope 下之 Seperate 則記錄頻道會分開。



● 邏輯分析設定：



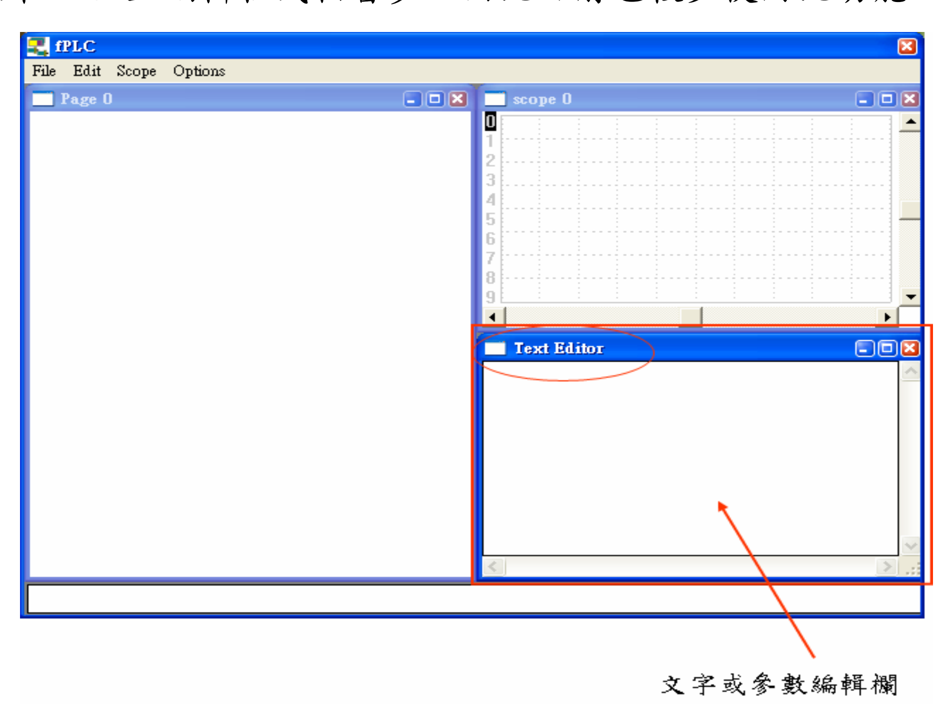


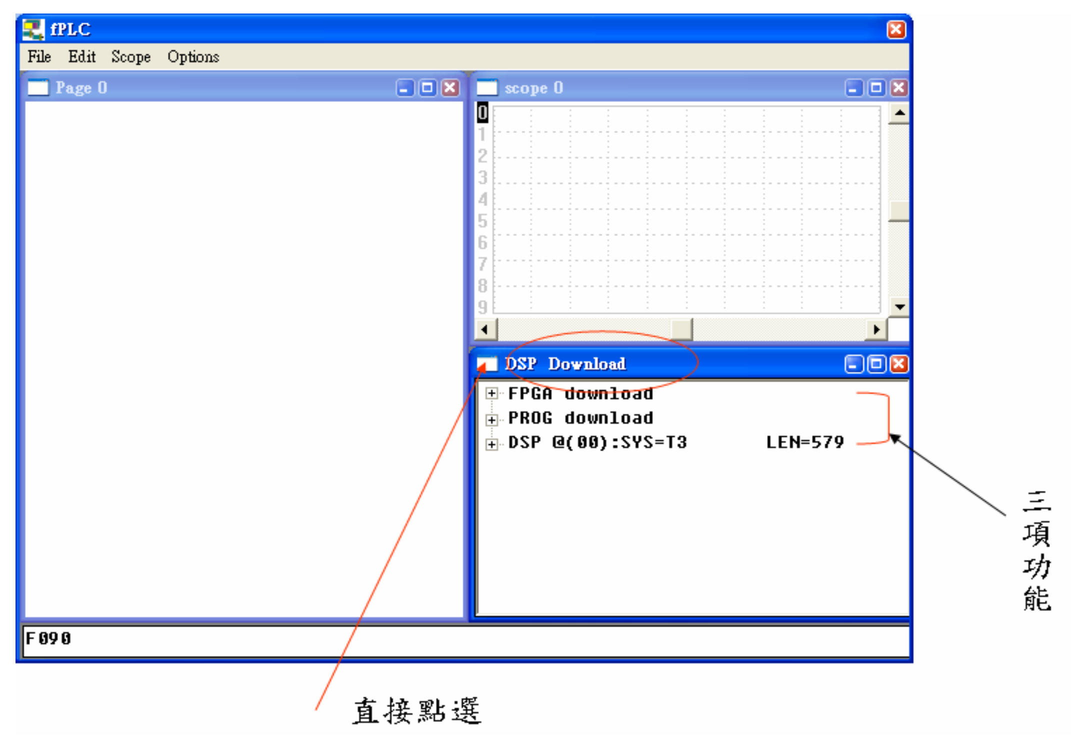
以邏輯分析功能觀測直流馬達之光電盤輸入與 PWM 輸出信號

1.3 FPLC 文字編輯欄說明

文字或參數編輯欄功能說明

當您進入 FPLC 時出現下面畫面，在右下邊功能欄我們稱為文字或參數編輯欄，並在該欄左上角出現 Text Editor 表示可以在此欄撰寫程式功能，但由於在 PC 上編輯程式相當多，因此目前也較少使用此功能。





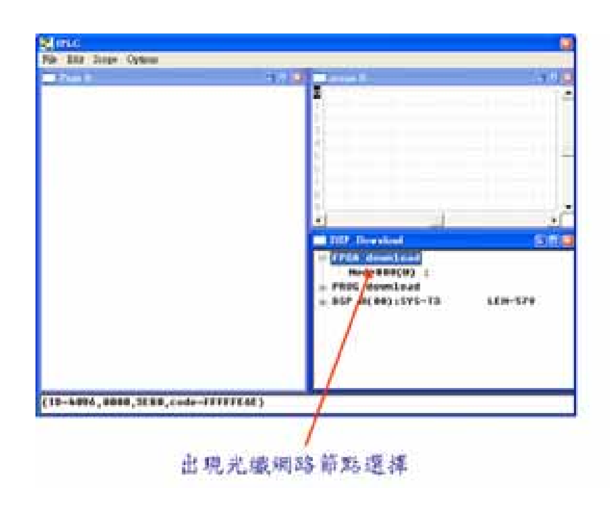
當點選該欄最左上角框框時，在該欄左上角出現 DSP Download 並顯示三項功能：

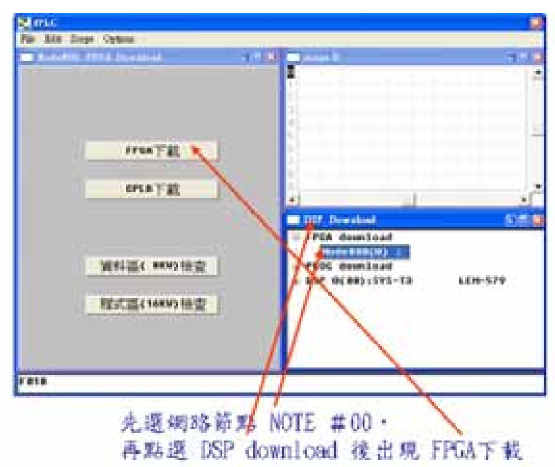
⊞ FPGA Download：下載各節點模組 FPGA 功能。

⊞ PROG Download：下載各節點模組系統連結程式

⊞ DSP @（00）SYS＝T3　　LEN＝579　：執行 DSP 設定 IO 與控制模組程式執行參數。

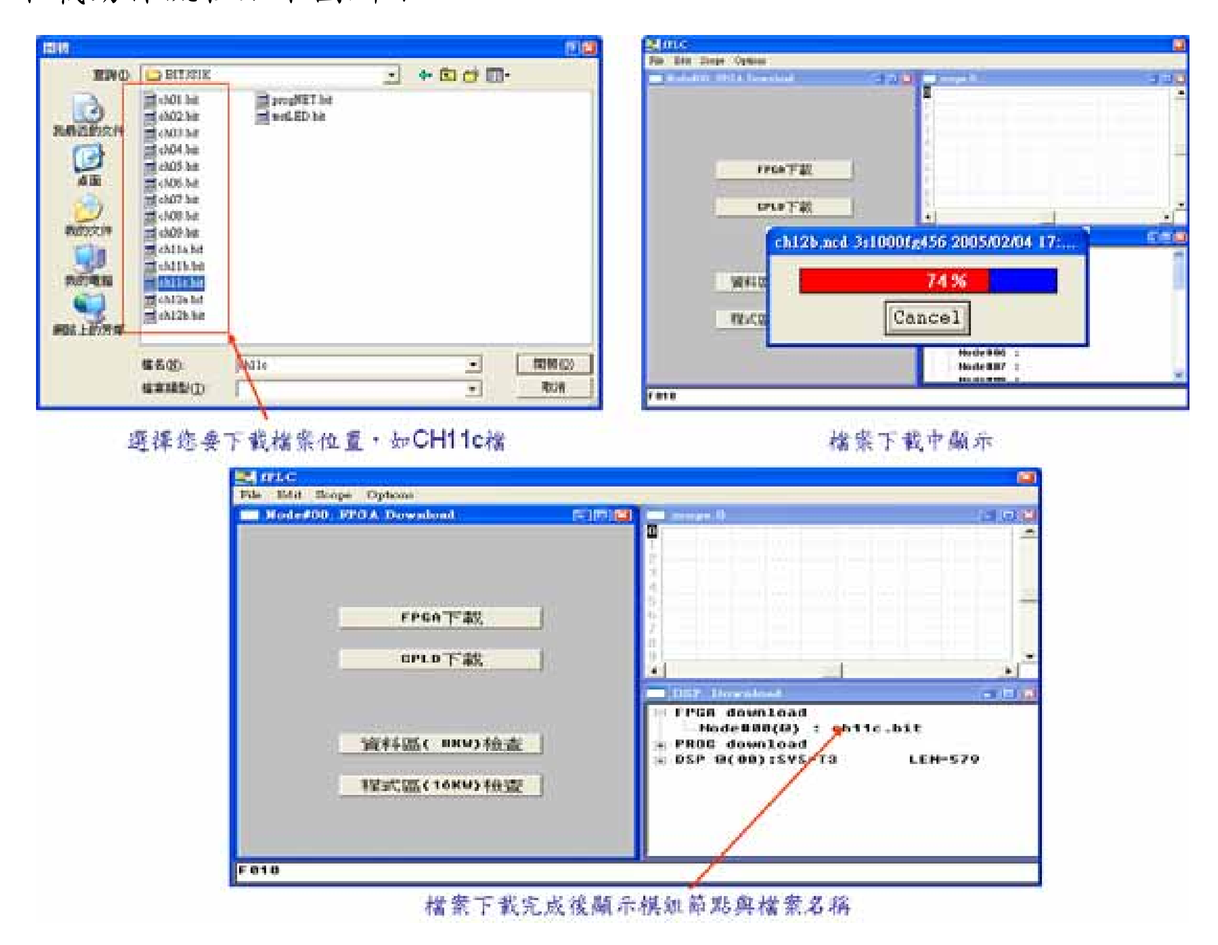
⊞ FPGA Download：下載各節點模組 FPGA 功能說明：





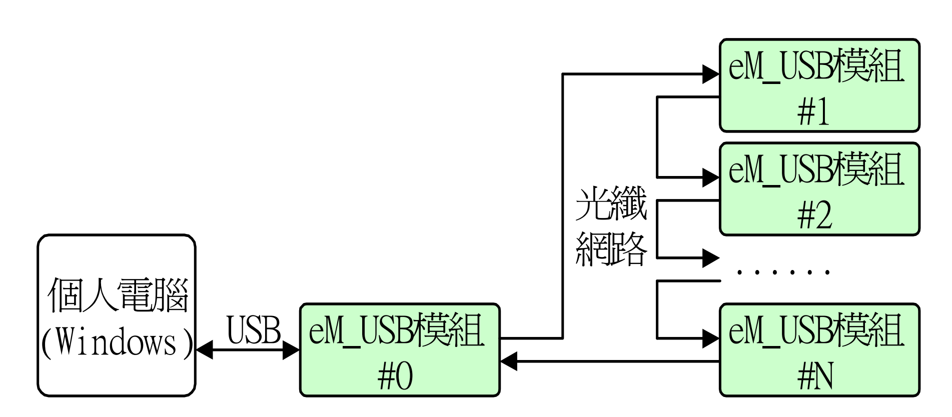
以滑鼠點選 ⊞ FPGA Download 中 ⊞ 出現網路模組節點＃00　Node @＃00：，只要在點選 DSP Download 在 FPLC 「人機界面或參數操控欄」出現 FPGA 下載畫面，如您使用光纖網路通訊 FPGA 發展模組，可選擇下載 FPGA 界面卡或 CPLD 界面卡。

下載動作流程如下圖所示：



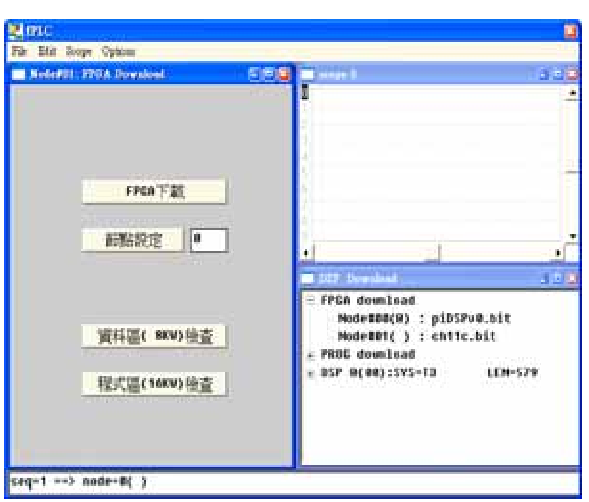
如您要使用多模組 FPGA 下載說明

首先利用光纖線，將 eM_USB＃0 模組光纖輸出埠接至 eM_USB＃1 模組光纖輸入埠，以此類推，最後再將最後 eM_USB＃N 模組光纖輸出埠接至 eM_USB＃0 模組光纖輸入埠，配置如附圖方式：



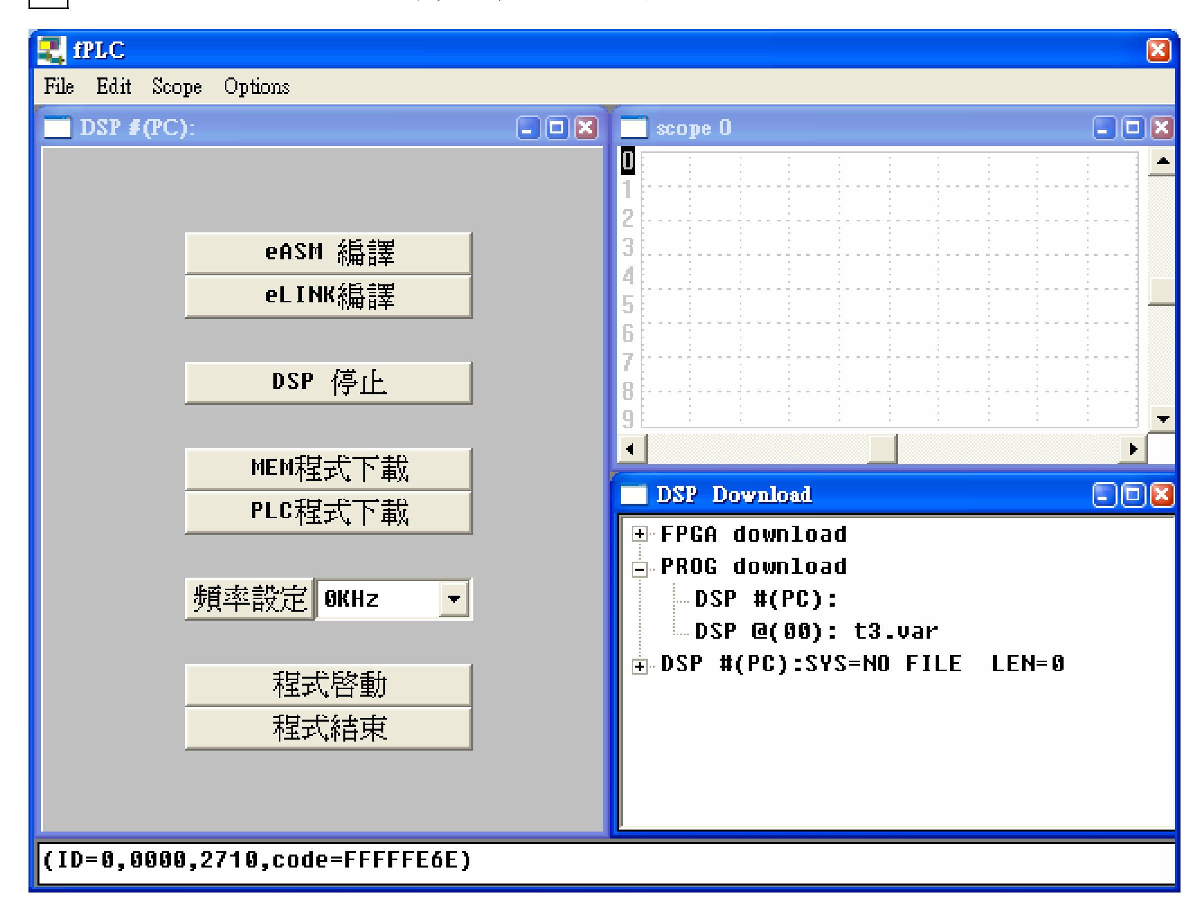
配置完成後，進入 FPLC 系統，首先必須先將 Node#00 下載在testASM 子目錄裡 piDSPv0.bit 檔，讓 eM_USB＃0模組 FPGA 具備 DSP 功能，才能執行光纖網路通訊，操作如下：





當 Node#00 已具備 DSP 功能時使用網路檢查功能檢查所有已經網路連線節點，並顯示所有節點號碼，此時您就可以選擇分別下載至各模組 FPGA 檔 *.bit ，假設您 Node#00 也要使用其他 FPGA 檔案也可以下載。

┼ PROG Download：下載各節點模組系統連結程式說明：



首先將滑鼠點選 PROG Download 在點選 DSP Download 在 FPLC 「人機界面或參數操控欄」出現上圖功能操控畫面，各功能說明如下：

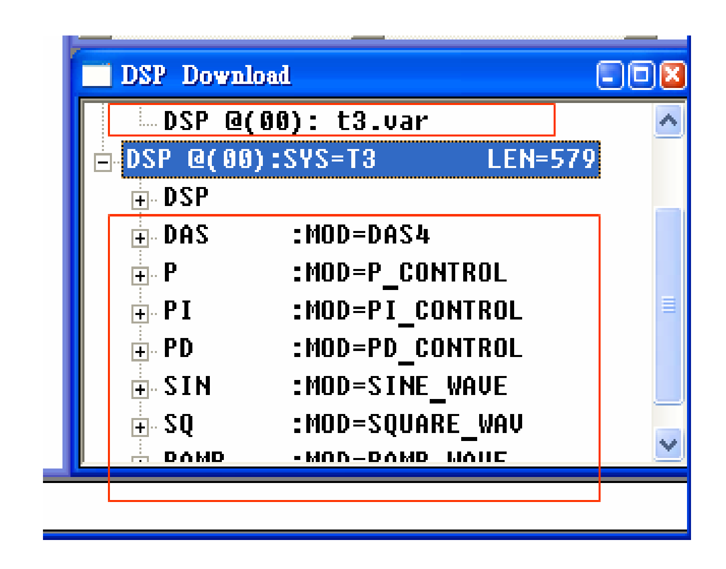
eASM 編譯：將撰寫好的 DSP 控制組合語言 *.ASM 編譯成模組程式 *.MOD

eLink 編譯：將撰寫好的系統連結檔 *.SYS 編譯成機械碼 *.MEM 和 變數檔 *.VAR 。

MEM 程式下載：下載系統連結檔 *.SYS 編譯成機械碼 *.MEM 和 變數檔 *.VAR 檔，先在 PROG Download 下點選節點模組再使用 MEM 程式下載您需要之 *.MEM ，完成後該節點模組會出現該檔案變數檔 *.VAR。

┼ DSP @（00）SYS＝T3　　LEN＝579　：執行 DSP 設定IO 與控制模組程式執行參數。

當您展開 ⊞ 時出現下面畫面：

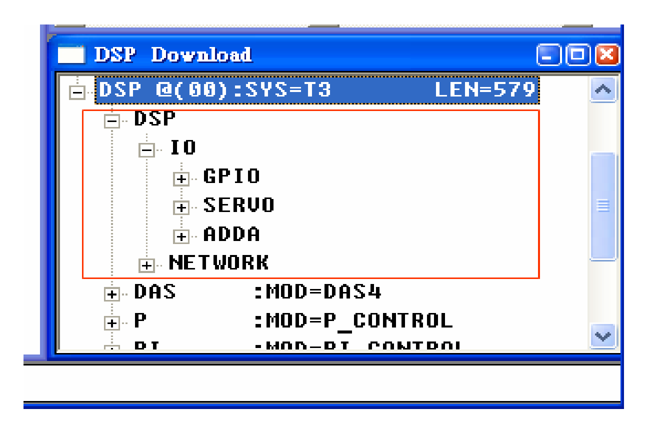


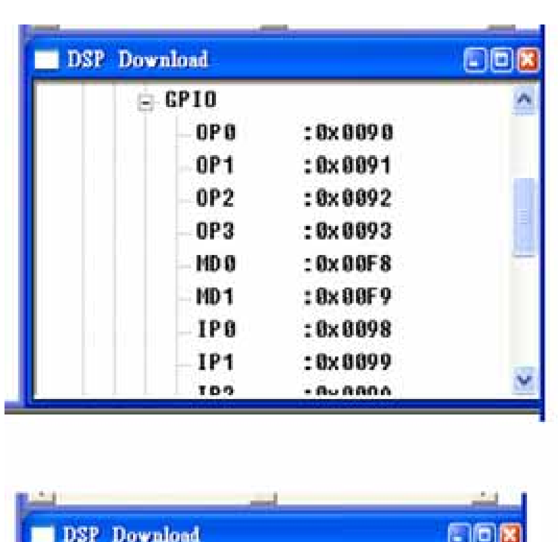
當您進入 FPLC 系統時 自動載入 t3.var ，其對應系統連結程式為

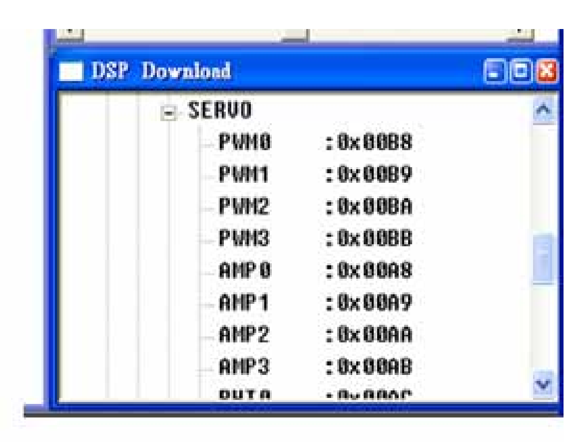
t3.sys

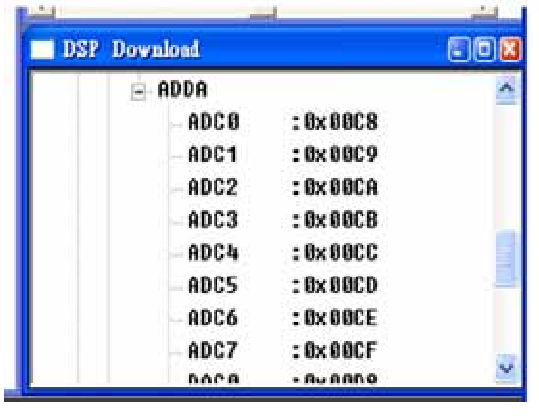
@Module	DAS	DAS4.mod;
@Module	P	P_control.mod;
@Module	PI	PI_control.mod;
@Module	PD	PD_control.mod;
@Module	SIN	SINE_wave.mod;
@Module	SQ	SQUARE_wave.mod;
@Module	RAMP	RAMP_wave.mod;

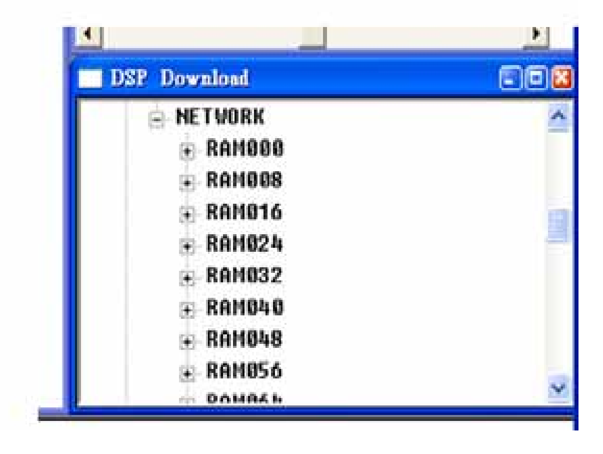
這時 FPLC 會將 t3.sys 宣告各程式模組顯示在此欄。
另外 DSP 是表示 FPLC 配合 piDSP 執行控制程式，所以同時也監控 DSP 使用 IO 與 RAM 相關資料



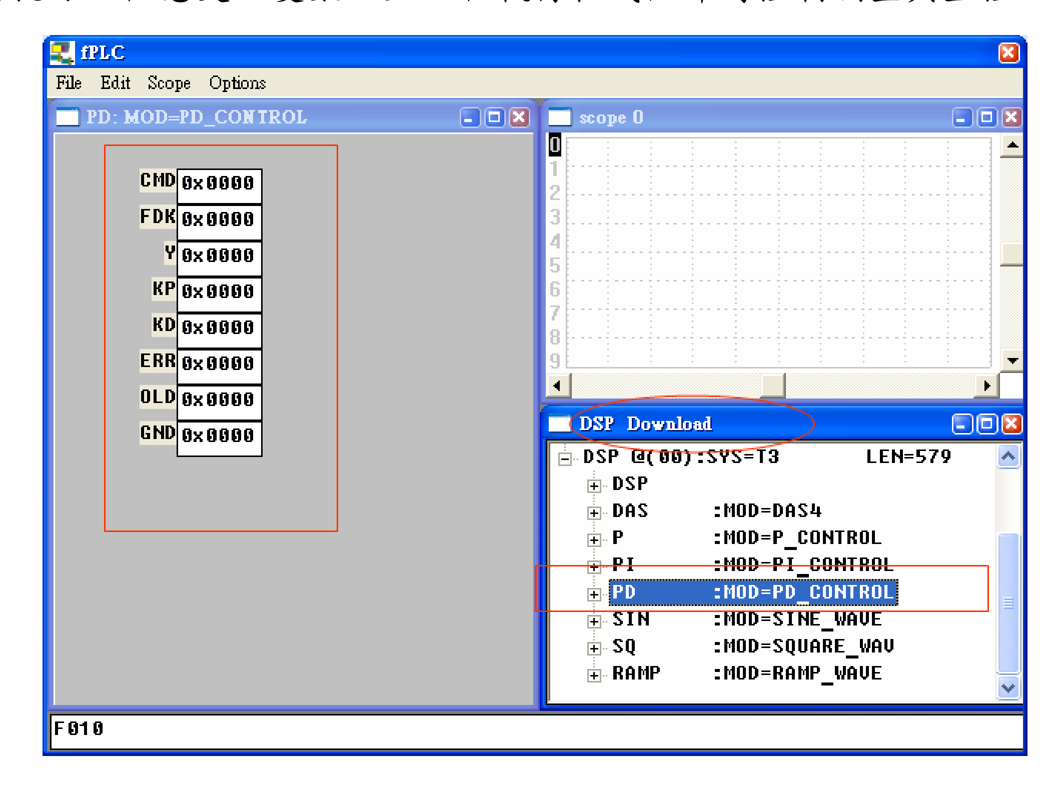








只要您以滑鼠點選該程式模組，再至 DSP Download 以滑鼠單擊，在 FPLC 「人機界面或參數操控欄」出現該模組所有宣告之變數欄，此時您就可以任意更改變數，並配合執行程式做即時控制調整與監控。



2.FPLC 操控畫面設計

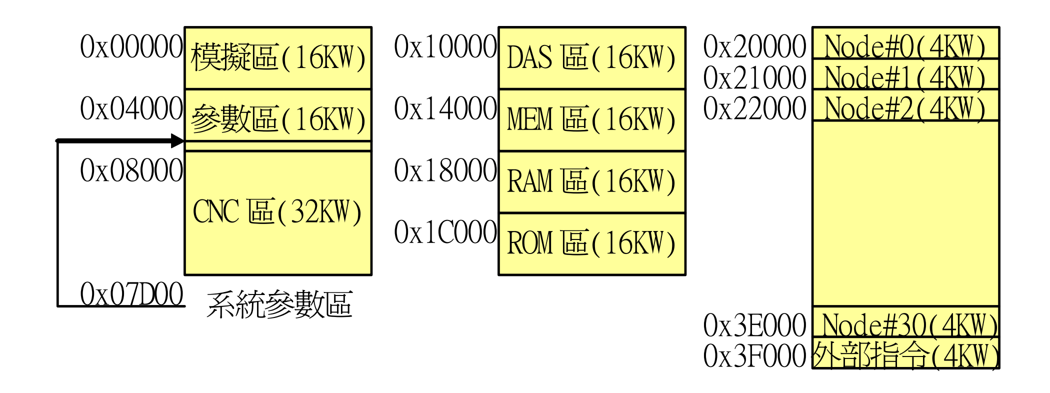
基本上 FPLC 可由兩種方式規劃操控畫面：

一、為直接撰寫 *.PLC 檔，在 FPLC 環境下 File 下 Program 指令呼叫該檔案就可以直接控制。

二、為使用 CAIDRW 規劃圖形監控畫面 *.CAI ，再搭配 *.PLC 檔，並在 FPLC 環境下 File 下 Setup 指令呼叫該檔案就可以操控漂亮人機圖控系統。

2.1 PLC 指令說明

公用記憶體配置



公用記憶體的空間配置如下：

其中：

1. PLC_data[]佔用256K-word的空間，前面64K-word位址配置如下：
   0x00000~0x01FFF(8KW) ：對應到DSP模擬器中的8K-word資料區
   0x02000~0x03FFF(8KW) ：對應到DSP模擬器中的8K-word程式區
   0x04000~0x07CFF(16KW)：可讓操作者到自由運用的參數區
   0x07D00~0x07FFF(768W)：對應到系統參數區
   0x08000~0x0FFFF(32KW)：對應到CNC控制用的參數區，目前尚未使用。
2. 其他部分的空間配置如下：
   0x10000~0x13FFF(16KW)：對應到示波器的DAS頻道和頁面控制。
   0x14000~0x17FFF(16KW)：對應到DSP中記憶體的上下載處理區
   0x18000~0x1BFFF(16KW)：對應到DSP中外部RAM的上下載處理區
   0x1C000~0x1FFFF(16KW)：對應到DSP中外部ROM的上下載處理區
   0x20000~0x3EFFF(4KW*31)：對應到31組DSP的4KW變數區
   0x3F000~0x3FFFF(4KW) ：對應到外部C程式的指令控制區

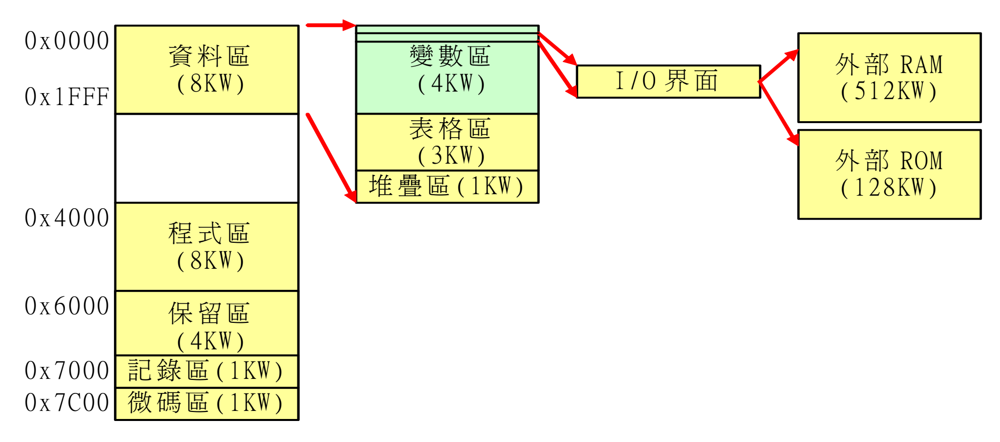
就操作者而言，可獨立運用的記憶體空間包括：

分類	資料	使 用 說 明
PLC區	PLC_data[256KW]	所有的公用記憶體空間，但是只有前面64KW才用PLC_data來直接讀寫
DAS區	DAS_data[16KW]	示波器控制用的資料蒐集區
MEM區	MEM_data[16KW]	DSP中記憶體的上下載處理空間
RAM區	RAM_data[16KW]	DSP中外部RAM的上下載處理空間
ROM區	ROM_data[16KW]	DSP中外部ROM的上下載處理空間
NET區	NET_data[31][4KW]	31組DSP變數區的對應空間
CMD區	PLC_command[4KB]	外部C程式的指令下達區
STS區	PLC_echo[4KB]	外部C程式的指令執行結果

DSP記憶體的配置

DSP核心中的記憶體配置如下：

其中：



1.DSP核心佔用32KW的記憶體空間，其中分成幾個部分：
  資料區(8KW)：佔用0x0000~0x1FFF，包括變數區、表格區和堆疊區。
  程式區(8KW)：佔用0x4000~0x5FFF，儲存機械碼。
  保留區(4KW)：佔用0x6000~0x6FFF，保留給影像處理等彈性用途使用。
  記錄區(1KW)：佔用0x7000~0x73FF，儲存硬體DAS資料。
  微碼區(1KW)：佔用0x7C00~0x7FFF，儲存微控碼表格。
  RAM區(>64KW)：外部的RAM記憶體，分成數頁，每頁以16KW爲單位作讀寫。
  ROM區(>64KW)：外部的FlashROM ，分成數頁，每頁以16KW爲單位作讀寫。

2.資料區中又可細分爲下面幾個部分：
  變數區(4KW)：佔用0x0000~0x0FFF，包括公用變數、模組變數和I/O界面。
  表格區(3KW)：佔用0x1000~0x1BFF，儲存表格和軟體DAS區。
  堆疊區(1KW)：佔用0x1C00~0x1FFF，系統堆疊區的位置。

3.I/O界面只佔用變數區的一小部分，由其中的幾個特殊界面來讀寫外部的RAM和FlashROM，所以外部RAM和FlashROM並未佔用32KW的記憶體空間。

當fPLC對DSP和新作監控時：

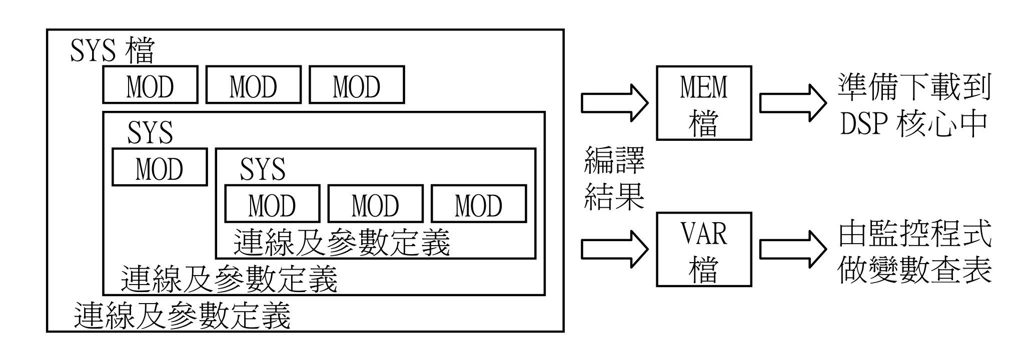
1.只有變數區(4KW)可以直接監控，對應到公用變數區中31組的NET區(4KW)。

2.其他部分無法直接監控，但可利用M指令做上載或下載處理。

3.當fPLC的操作者要監控DSP核心時，
  A.首先指定DSP節點，可選擇(0~30)中的任一個節點。
  B.讀寫R(0~4095)即可直接讀寫PC端的NET區，也同時對應到DSP端的變數區。
  C.讀寫R(>4095)的位址將會讀寫PC端的PLC區(512KW)，但不會讀寫DSP核心。
  D.可以讀寫PC端的MEM區(16KW)，再以M指令將資料上載或下載至DSP核心端。

變數格式定義

由於系統程式採取階層式的系統架構，在編譯和連結後如果要正確的監控其變數值，就要直接的處理變數命令。系統程式的架構圖如下：



其中的VAR檔就是記錄其中的變數結構。當fPLC讀取VAR檔而取得系統程式的變數結構後，就可接受變數命令的輸入。變數的格式定義如下：

.	根系統(即所編譯的SYS檔案)
var	在預設模組下的變數
..var	高一層模組下的變數
sys.var	低一層模組下的變數
.sys.sys.sys.var	從根系統算起的任意模組下的變數

在命令模式中可以設定預設模組，指令如下：

MODULE <var>	：變更預設模組，其中<var>爲任意組合的變數定義
MODULE	：顯示目前的預設模組

一旦有了預設模組，如果在變數中不特別指定，就是該模組下的變數。不但可以省下長串的文字敘述，也可以增加程式的可讀性。所有D/S/F/C指令中的參數都可選擇常數或變數格式，說明如下：

0x1234,1234或-1234	：可以直接以數字型式設定參數，可選擇10進位或16進位
var	：以變數型式設定參數，代表var的內容
*var	：以變數型式設定參數，以var內容爲指標所指向位址的內容
&var	：以變數型式設定參數，代表var的位址

其中：

1. 當參數爲<addr>時，以變數的位址來設定<addr>參數。
2. 當參數爲<data>時，以變數的內容來設定<data>參數。

Dump指令

D指令是用來顯示對映記憶體的內容，其格式包括：

```
D [area] <addr> <n>; //**顯示記憶體中的N筆資料
D [area] <addr>;     //**顯示記憶體中的1筆資料
D;                   //**顯示記憶體中的下8筆資料
D <addr>  0;         //**顯示指定位址所對映的變數名稱
```

其中：

1.D指令是用來顯示記憶體的內容，在使用之前必須先選擇DSP編號，其中
   DSP=0　　：光纖網路中的主節點。
   DSP=1~30：光纖網路中的輔節點。
   DSP=其他：指向PC端的公用記憶體。
2.對於PC端的公用記憶體，可選擇：
   PLC區：爲預設值，有效位址範圍爲(0x00000~0x3FFFF)的256KW空間。
   DAS區：當<area>=DAS時，有效位址範圍爲(0x0000~0x3FFF)的16KW空間。
   MEM區：當<area>=MEM時，有效位址範圍爲(0x0000~0x3FFF)的16KW空間。
   RAM區：當<area>=RAM時，有效位址範圍爲(0x0000~0x3FFF)的16KW空間。
   ROM區：當<area>=ROM時，有效位址範圍爲(0x0000~0x3FFF)的16KW空間。
3.對於光纖網路中的DSP節點，可選擇：
   變數區：有效位址範圍爲(0x0000~0x0FFF)的4KW空間。
   任何超過這個範圍的位址，都會自動轉向PC端中的PLC區。
4.可利用[area]來指定DAS/MEM/RAM/ROM，當未指定<area>時，
   若節點選擇DSP=0~30，則預設區域爲DSP端的變數區，
   否則預設區域爲PC端的PLC區。
5.也可用D指令來顯示對映位址的變數名稱，例如：
   D <addr> 0;
   如果該位址存在變數名稱，則會自動找出變數名稱做顯示。

Set指令

S指令是用來設定對映記憶體的內容，其格式包括：

```
S [area] <addr> <d>;             //**設定記憶體中的1筆資料
S [area] <addr> <d1> <d2> ..;    //**設定記憶體中的N筆資料，其中N<=10
```

其中：

1.S指令用來設定記憶體的內容。和D指令一樣，使用之前必須先選擇DSP編號。

2.對於光纖網路中的DSP節點，可選擇：

變數區：有效位址範圍爲(0x0000~0x0FFF)的4KW空間。

任何超過這個範圍的位址，都會自動轉向PC端中的PLC區。

3.可利用[area]來指定DAS/MEM/RAM/ROM，當未指定<area>時，

若節點選擇DSP=0~30，則預設區域爲DSP端的變數區，

否則預設區域爲PC端的PLC區。

S指令是批次檔中最常用的指令，通常用來設定模組程式中各項參數的內容。

Fill指令

F指令是用來充填對映記憶體的內容，其格式包括：

```
F [area] <addr> <len> <d1> <d2> ..; //**充填記憶體中的N筆資料，N<=10
```

其中：

1.F指令用來充填記憶體的內容。和D指令一樣，使用之前必須先選擇DSP編號。

2.對於光纖網路中的DSP節點，可選擇：

變數區：有效位址範圍為(0x0000~0x0FFF)的4KW空間。

任何超過這個範圍的位址，都會自動轉向PC端中的PLC區。

3.可利用[area]來指定DAS/MEM/RAM/ROM，當未指定<area>時，

若節點選擇DSP=0~30，則預設區域為DSP端的變數區，

否則預設區域為PC端的PLC區。

4.F指令中的充填範圍是從<addr>開始的<len>長度，例如：

```
F 0    100  ...;     //充填0~99之間的100-word
F 1000 2000 ...;     //充填1000~2999之間的2000-word
```

5.F指令是根據指定的N筆資料做依序的充填動作，當N筆資料小於充填範圍時，剩餘空間以循環碼做充填。例如：

```
F 0 100 1 2 3;       //充填內容依序為1/2/3/1/2/3/1/2/3/1/...
F 0 100 0xffff;      //全部範圍都以0xffff做充填
```

Compare指令

C指令是用來比較對映記憶體的內容，其格式包括：

```
C [area] <addr> <len> <d1> <d2> ..; //**比較資料區中的N筆資料
```

其中：

1. C指令用來比較記憶體的內容。和D指令一樣，使用之前必須先選擇DSP編號。
2. 對於光纖網路中的DSP節點，可選擇：
   變數區：有效位址範圍為(0x0000~0x0FFF)的4KW空間。
   任何超過這個範圍的位址，都會自動轉向PC端中的PLC區。
3. 可利用[area]來指定DAS/MEM/RAM/ROM，當未指定<area>時，
   若節點選擇DSP=0~30，則預設區域為DSP端的變數區，
   否則預設區域為PC端的PLC區。
4. C指令的比較範圍是從<addr>開始的<len>長度，例如：
   ```
   C 0     100  ...;     //比較0~99之間的100-word
   C 1000  2000 ...;     //比較1000~2999之間的2000-word
   ```
5. C指令是根據指定的N筆資料做依序的比較動作，當N筆資料小於比較範圍時，剩餘空間以循環碼做比較。例如：
   ```
   C 0 100 1 2 3;        //比較內容依序為1/2/3/1/2/3/1/2/3/1/...
   C 0 100 0xffff;       //全部範圍都以0xffff做比較
   ```
6. 在指令格式上，C指令和F指令的格式完全相同，通常也是兩者搭配一起使用。例如：先以F指令充填記憶體，再以C指令來比較記憶體的內容，就可檢查對映的記憶體有無問題。

Move指令

M指令是用來執行記憶體上下載的工作，其格式包括：

```
M UP   [node] <addr> <len>;        //**執行資料區或程式區的上載工作
M DOWN [node] <addr> <len>;        //**執行資料區或程式區的下載工作
M PAGE <code>;                     //**設定RAM/ROM的頁數管理
M UP   [node] <addr> <len> RAM;    //**執行外部RAM的上載工作
M DOWN [node] <addr> <len> RAM;    //**執行外部RAM的下載工作
M UP   [node] <addr> <len> ROM;    //**執行外部FlashROM的上載工作
M DOWN [node] <addr> <len> ROM;    //**執行外部FlashROM的下載工作
M <sN> <sA> <dN> <dA> <len>;       //**PC端公用記憶體的內部移動
```

其中：

1.M UP ..指令用來執行上載工作，而M DOWN ..指令用來執行下載工作。
2.對DSP核心中的資料區和程式區作上下載工作時，
　　以[node]來選擇DSP核心的網路編號，當node不存在時即爲預設節點。
　　以<addr>和<len>來選擇記憶體範圍。
3.當記憶體範圍超過0x2000的位址時，定義成程式區的記憶體上下載處理，否則定義成資料區的記憶體上下載處理。
4.對外部RAM/ROM的處理時，必須先設定PAGE參數之後才執行上下載工作。一旦PAGE參數設定之後，每次可讀寫16KW的範圍。
5.另外也可直接對PC端的公用記憶體作資料搬移的動作。包括RAM區、ROM區、DAS區和MEM區。

對於PAGE參數的定義如下：

1.PAGE參數爲16-bit，其中
　b15~8：固定爲"11111000"。
　b7~4 ：頁數選擇，可選擇0~15。
　b3 　：記憶體選擇，1=FlashROM，0=RAM。
　b2~0 ：固定爲"000"。
2.下面是幾個選擇範例：
　M PAGE 0xf800;　選擇RAM的PAGE#0
　M PAGE 0xf8f0;　選擇RAM的PAGE#15
　M PAGE 0xf808;　選擇FlashROM的PAGE#0
　M PAGE 0xf8f8;　選擇FlashROM的PAGE#15
3.在eM_USB模組中，RAM的容量爲256KW，可選擇16頁(0~15)，而FlashROM的容量爲64KW，可選擇4頁(0~3)。
4.FlashROM的最後一頁固定爲DSP核心的開機啓動程式區(16KW)，對映到變數區(4KW)、表格區(3KW)、微碼區(1KW)和程式區(8KW)。

Run指令

RUN指令是用來執行DSP核心的直接控制工作，其格式包括：

```
RUN DSP   <n>;        //**選擇預設的DSP核心
RUN RESET;            //**凍結DSP核心，準備下載程式
RUN <filename>;       //**將編譯好的組合語言程式下載至DSP核心中
RUN <1kHz>;           //**選擇@timer程序的中斷頻率，可選擇
                          0kHz/1kHz/2kHz/4kHz/8kHz/16kHz/32kHz
RUN FREQ <n>;         //**選擇@timer程序的中斷頻率爲nHz
RUN START;            //**啓動DSP核心開始執行組合語言程式
RUN STOP;             //**結束DSP核心的程式執行工作
RUN STEP <n>;         //**當使用DSP模擬器時，用來執行單步模擬的工作
```

其中：

1. DSP核心中需要下載的程式部分共計16K-word，包括：
   變數區：0x0000開始的4K-word
   表格區：0x1000開始的3K-word
   程式區：0x4000開始的8K-word
   微碼區：0x7C00開始的1K-word
2. 一旦DSP核心開始運作，程式區和微碼區就不准PC端作讀寫。因此要下載程式之前，必須先以RUN RESET指令將DSP核心凍結，之後才准作上下載工作。
3. 下載程式的細部動作如下：
   A.將編譯好的MEM檔讀入，並用以設定16K-word的MEM區。
   B.固定讀入微控碼檔案(mCode.cod)，設定MEM區中的對映空間。
   C.將MEM區的16K-word內容，以M指令下載至DSP核心之中。
   D.將編譯好的VAR檔讀入以設定fPLC內部的變數表格。一旦變數表格設好，就可用變數型式來選擇變數的記憶體位址。
4. 下載程式完畢後，還可用S指令來設定模組程式中各項參數。
5. 執行前的動作依序如下：
   A.設定@timer程序的中斷頻率，
   B.啓動DSP核心。
6. 在DSP核心啓動之後，可利用RUN STOP指令叫停，也可再用RUN START指令重新啓動。
7. 由於fPLC核心可面對超過30顆的DSP核心，所以必須先以RUN DSP指令來選擇所指定的DSP核心，才能用D/S/F/C等指令做處理。一次只能處理一顆DSP核心。
8. 因爲每顆DSP核心中的組合語言程式可能都不一樣，所以在選擇不同的DSP核心之後，對映的變數格式和位址也會相對的跟著改變。

Echo指令

ECHO 指令用來控制變數的顯示方式，其格式包括：

```
ECHO ON;           //**設定ECHO=ON，這時會顯示直接指令的執行結果
ECHO OFF;          //**設定ECHO=OFF，將不會顯示任何執行結果
ECHO <X/U/D>;      //**選擇變數的顯示格式，包括
                        X:16進位，U:正整數，D:整數(有正負號)
       //***** 下面指令只准用在PLC檔中，作爲畫面的顯示用 *****
ECHO CLEAR;        //**清除畫面
ECHO "message";    //**顯示一行文字串
ECHO DISPLAY;      //**執行顯示動作
```

其中：

1. 直接指令在執行時，都會在直接指令視窗中以文字串型式顯示執行結果。但是在批次指令執行時，往往不需要顯示每一筆的執行結果。這時可以用ECHO ON指令和ECHO OFF指令來決定是否要作顯示。
2. 另外在CAI視窗中，變數顯示格式若以X格式來定義，則可臨時以ECHO指令選擇不同的顯示格式。
3. 在PLC檔中可以用ECHO指令來控制CAI視窗的顯示，提供一個比較簡單的選擇，不用撰寫CAI檔就能執行操作畫面。

Key指令

KEY指令是用來定義功能鍵的作用，其格式包括：

```
KEY <page> <n> "name" "command";    //**定義某個的功能鍵<Fn>
KEY <page> 11  "name";              //**定義整頁的名稱
KEY <page>  0;                      //**清除整頁的功能鍵
KEY <page>;                         //**直接選擇頁數
```

其中：

1.功能鍵共有100組，分成10頁來處理，每頁都包括10組功能鍵。

2.10組功能鍵分別對應<F1>~<F10>，而<F11>和<F12>用來選擇頁數。其中

<F11>：頁數循環遞減

<F12>：頁數循環遞增

3.頁數標示範圍為(1~10)，而功能鍵標示範圍也是(1~10)。所以

KEY <page> <n> "name" "command"

就是用來定義某個功能鍵的內容，其中

page：指定頁數，範圍為(1~10)

n　：指定功能鍵，範圍為(1~10)

name：功能鍵上的名稱顯示，通常為4位的中文，或是8位的英文

command：按下功能鍵時所要執行的批次指令內容

4.由於頁數共有10頁，每頁除了頁數顯示之外，還可以顯示不同名稱做區隔。這時可用

KEY <page> 11 "name"

來設定該頁的名稱。

5.若要清除原先設定的功能鍵，可用

KEY <page> 0

來清除該頁的設定，一次只能清除一頁。

另外KEY指令還有兩項特殊功能，表列如下：

```
KEY <page> <-n>;              //**直接執行功能鍵<Fn>
KEY -1;                       //**以功能鍵內容來設定表單
KEY  0;                       //**設定標準格式的英文表單
KEY -9;                       //**設定標準格式的中文表單
```

其中：

1.可以直接執行功能鍵，定義如下：

KEY <page> <-n>

即可模擬某頁中的<Fn>動作。

2.也可以將功能鍵換到畫面上方的表單中顯示，定義如下：

KEY -1;

即可更新表單的定義。若要還原成標準表單則可用(KEY 0)或(KEY -9)指令。

Module指令

MODULE指令是用來簡化變數名稱的敘述工作，其格式包括：

```
MODULE <name>;         //**指定預設模組的名稱
MODULE;                //**顯示預設模組的名稱
```

變數的格式定義如下：

.	根系統(即所編譯的SYS檔案)
var	在預設模組下的變數
..var	高一層模組下的變數
sys.var	低一層模組下的變數
.sys.sys.sys.var	從根系統算起的任意模組下的變數

預設模組可以簡化變數的定義型式，不用每筆變數都從根系統開始敘述。有了預設模組，如果在變數中不特別指定，就是該模組下的變數。不但可以省下長串的文字敘述，也可以增加程式的可讀性。例如：

1. 首先設定預設模組，指令如下：

   MODULE .fn1;

2. 之後即可指定變數名稱，

   x　　即代表.fn1.x，而

   y　　即代表.fn1.y。

Net指令

NET 指令是用來執行光纖網路的管理工作，其格式包括：

```
NET;              //**顯示目前的光纖網路狀態
NET <seq>;        //**顯示該序號之下的網路節點
NET <seq> <n>;    //**設定該序號之下的網路節點
```

其中：

1.光纖網路上的每個DSP核心都有兩種編號：

網路序號：依據光纖的連接順序而得的自然序號，其中主節點的序號為0，從主節點接出的輔節點序號為1，之後再接出的輔節點序號為2，依此類推。

網路編號：自行定義的編號，主節點固定為0，而輔節點可自由選擇1~30中的任何編號，只要彼此不重複即可。

2.所以需要兩種編號的原因如下：

A.就工作需求而言，我們希望編號是固定的，例如
X軸伺服控制的網路節點為'X'，而
Y軸伺服控制的網路節點為'Y'。
即使光纖連接方式更改，也不應更改網路節點的編號。

B.網路節點的編號需要由PC端來設定，一旦設定之後就固定了。

C.但是在網路節點設定之前，該如何選擇網路上的某一節點呢？這時就要根據網路序號來指定了。

3.換句話說：

A.先根據網路序號來選擇網路節點，並設定該網路節點的編號。

B.一旦網路節點的編號固定後，就不再使用網路序號，而以網路編號來指定網路節點。

C.之後即使光纖連線切換也不會變更網路節點的編號。

在執行 RUN DSP <n> 指令來選擇網路節點時，<n> 就是網路編號，而不是網路序號。必須特別注意。

Prog指令

PROG 指令是用來執行批次檔（PLC檔）的啟動工作，其格式包括：

```
PROG <filename.PLC>;        //**執行指定的批次檔
```

其中：

1. PLC檔是一般的文書檔，內容是將一連串的直接指令撰寫在一個檔案中，而在執行時則依序執行其中的每一個直接指令。
2. PLC檔中還可以呼叫其他的PLC檔，產生階層式的呼叫結構。等被呼叫的PLC檔執行完畢後，再從這個PLC檔的下一個直接指令繼續的執行，直到所有的指令都執行完畢。

PLOT指令

示波器的控制指令包括：

```
PLOT select <n>;                        選擇示波器的頁數
PLOT xUnit      <us> <x0>;              選擇X軸的單位和平移量
PLOT yUnit <ch> <dy> <y0>;              選擇Y軸的單位和偏移量
PLOT trigger <mode> <slope> <chx> <level> "bits"; 設定觸發模式
PLOT scope <mode> <ch1> <ch2> <ch3> <ch4>;        設定顯示模式
```

FPGA指令

FPGA 指令是用來執行 FPGA 電路的下載工作，其格式包括：

```
FPGA <filename> <n>;        //**執行第<n>個FPGA的下載工作
```

其中：

1. 在FPGA電路下載時，會先清除原先的FPGA電路。由於DSP核心本身就是FPGA中的邏輯電路，所以這時DSP核心已經無法正常工作。
2. 網路節點的編號是DSP核心中的一項功能，由於DSP核心已不能工作，所以網路節點也無法定義。
3. 換句話說，在下載FPGA電路時，必須以網路序號來指定網路節點，而不是以網路編號來指定網路節點。
4. 等FPGA電路下載完成後，還需以NET指令來設定網路編號，之後這顆DSP核心才算是正常工作。

DAS指令

DAS的控制指令包括：

```
DAS <ch> <addr> <bias> <unit> "name" "unit"; 設定DAS頻道
DAS start;                    啟動DAS工作
DAS stop;                     停止DAS工作
```

Pause指令

PAUSE 指令是用來執行批次檔中的暫停動作，其格式包括：

```
PAUSE <n> "message";          //**暫停n秒之後再繼續執行
```

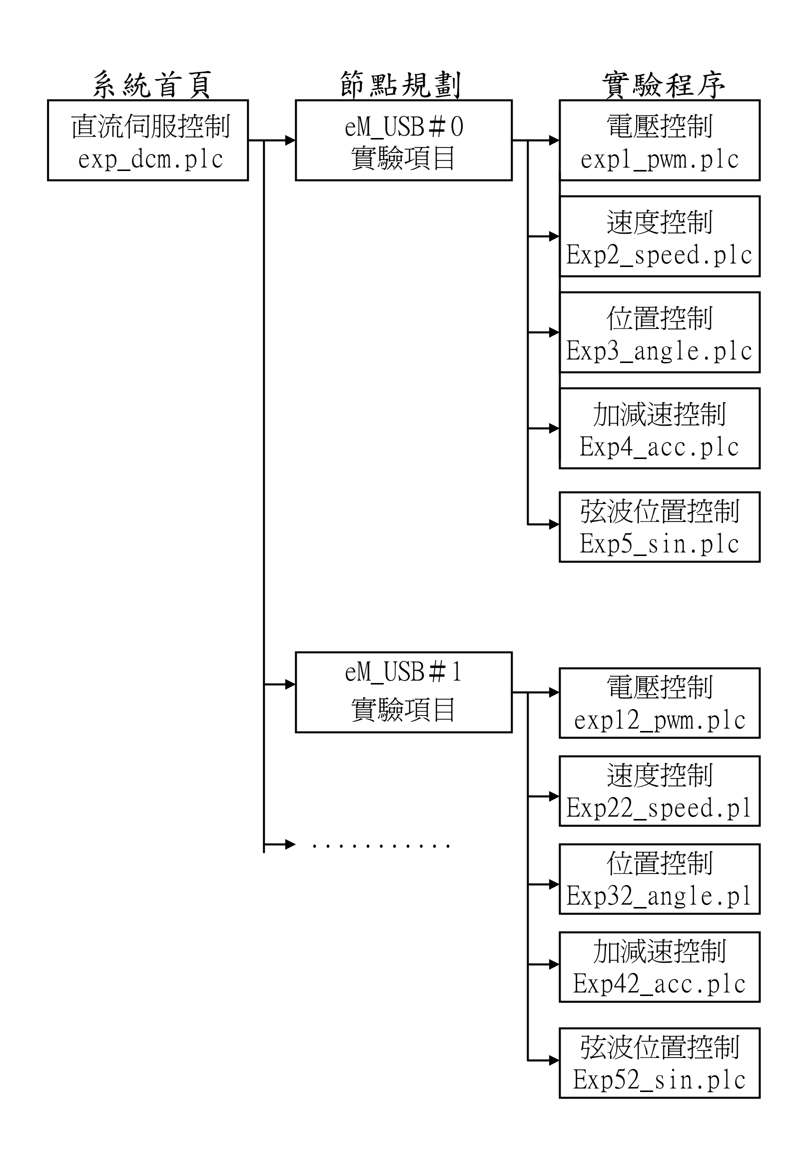
其中：

1.PAUSE指令以秒爲單位。
2.在暫停過程中可顯示"message"來指示暫停的原因。

2.2 PLC 實驗規劃

檔案結構規劃

PLC 檔是執行實驗程序的批次檔，用來導引實驗程序的進行。PLC 檔可以建立階層式的操作結構，經由各個 PLC 檔之間的相互呼叫，就可以串接成一個方便的操作環境。以 exp_dcm.plc 內容而言，整個檔案操作結構如下：



1. 首先建立一個最高層的系統首頁，代表整個直流伺服控制實驗的開始。
2. 在系統首頁中，可以選擇呼叫不同節點實驗項目。每一節點章節或選擇各項實驗。
3. 為了方便起見，eM_USB＃0 的 PLC 檔都以 exp*_ 做開頭，eM_USB＃1 的PLC檔都以exp*2_做開頭，依此類推。
4. 每一節點實驗中可以再細分成不同的實驗項目。以 eM_USB＃0 而例，分別為：

   exp1_pwm.plc: 實驗一：開環路的電壓控制
   exp2_speed.plc: 實驗二：閉環路的速度控制
   exp3_angle.plc: 實驗三：閉環路的位置控制
   exp4_acc.plc: 實驗四：加減速的位置控制
   exp5_sin.plc: 實驗五：正弦波的位置控制

   eM_USB＃1而例，分別為：

   exp12_pwm.plc: 實驗一：開環路的電壓控制
   exp22_speed.plc: 實驗二：閉環路的速度控制
   exp32_angle.plc: 實驗三：閉環路的位置控制
   exp42_acc.plc: 實驗四：加減速的位置控制
   exp52_sin.plc: 實驗五：正弦波的位置控制

5. 為了方便起見，每一節點實驗只有一個系統程式。testDCM.sys 是 eM_USB＃0 的系統程式， testDCM2.sys是 eM_USB＃1 的系統程式，依此類推。
6. 整個直流伺服控制實驗所需的所有程式，包括 PLC 檔、SYS 檔、ASM 檔等等，都放在同一個子目錄下，以方便管理。
7. 模組程式並不單屬於某個實驗，而是由所有實驗所分享，所以模組程式的名稱定義不受節點所限定。

實驗首頁設計

批次檔範例：exp_dcm.plc

```
echo off;
run stop;
echo clear;
echo "************************************";
echo "*          直流伺服控制實驗          *";
echo "************************************";
echo " ";
echo "實驗一. 開環路的電壓控制";
echo "實驗二. 閉環路的速度控制";
echo "實驗三. 閉環路的位置控制";
echo "實驗四. 加減速的位置控制";
echo "實驗五. 正弦波的位置控制";
echo display;

KEY 2 0;
KEY 2 1  "實驗一" "prog exp1_pwm.plc";
KEY 2 2  "實驗二" "prog exp2_speed.plc";
KEY 2 3  "實驗三" "prog exp3_angle.plc";
KEY 2 4  "實驗四" "prog exp4_acc.plc";
KEY 2 5  "實驗五" "prog exp5_sin.plc";
KEY 2 6  "實驗停止" "run dsp 0; run stop";
KEY 2 7  "DAS啟動" "das start";
KEY 2 8  "DAS停止" "das stop";
KEY 2 9  "首頁"   "run stop;prog exp_dcm.plc";
KEY 2 10 "電路下載" "prog download.plc";
KEY 2 11 "直流1";
KEY 2;

KEY 3 0;
KEY 3 1  "實驗一" "prog exp12_pwm.plc";
KEY 3 2  "實驗二" "prog exp22_speed.plc";
KEY 3 3  "實驗三" "prog exp32_angle.plc";
KEY 3 4  "實驗四" "prog exp42_acc.plc";
KEY 3 5  "實驗五" "prog exp52_sin.plc";
KEY 3 6  "實驗停止" "run dsp 1; run stop";
KEY 3 7  "DAS啟動" "das start";
KEY 3 8  "DAS停止" "das stop";
KEY 3 9  "首頁"   "run stop;prog exp_dcm.plc";
KEY 3 10 "電路下載" "prog download.plc";
```

```
KEY 3 11 "直流2";
KEY 3;
```

其中的設計重點包括：

1. 首先以echo指令顯示操作畫面，包括
   ```
   echo off;                    //**取消eTTY中的應答訊息顯示
   echo clear;                  //**清除整個畫面
   ```
2. 接著以echo指令顯示實驗標題畫面，
   echo "...";　如下面規劃畫面

```
echo "************************************";
echo "*            直流伺服控制實驗           *";
echo "************************************";
echo " ";
echo "實驗一. 開環路的電壓控制";
echo "實驗二. 閉環路的速度控制";
echo "實驗三. 閉環路的位置控制";
echo "實驗四. 加減速的位置控制";
echo "實驗五. 正弦波的位置控制";
```

在FPLC執行程式時，會根據您所規劃畫面順序顯示，最後以 echo display; 完成顯示指令。



3. 功能建設定：在 FPLC PLC 系統可設定 10 頁（0-9）功能鍵，每一頁可規劃 12 組功能設定鍵（0-11），如

KEY 2 1 "實驗一" "prog expl_pwm.plc";

表示：第 2 頁第 1 組功能顯示（實驗一），滑鼠或鍵盤功能鍵執行時便執行 expl_pwm.plc 實驗

KEY 3 3 "實驗三" "prog exp32_angle.plc";

表示：第 3 頁第 3 組功能顯示（實驗三），滑鼠或鍵盤功能鍵執行時便執行 exp32_ angle.plc 實驗

KEY 2 1	KEY 2 3	KEY 2 5	KEY 2 7	KEY 2 9	KEY 2 11
F1:實驗一	F3:實驗三	F5:實驗五	F7:DAS啓動	F9:首頁	P2:直流1
F2:實驗二	F4:實驗四	F6:實驗停止	F8:DAS停止	F0:電路下載	
KEY 2 2	KEY 2 4	KEY 2 6	KEY 2 8	KEY 2 10	KEY 2 0

KEY 3 1	KEY 3 3	KEY 3 5	KEY 3 7	KEY 3 9	KEY 3 11
F1:實驗一	F3:實驗三	F5:實驗五	F7:DAS啓動	F9:首頁	P3:直流2
F2:實驗二	F4:實驗四	F6:實驗停止	F8:DAS停止	F0:電路下載	
KEY 3 2	KEY 3 4	KEY 3 6	KEY 3 8	KEY 3 10	KEY 3 0

滑鼠點選一下功能鍵跳到下一頁，滑鼠點選 P3 位置一下功能鍵跳到上一頁。

```
4.run stop                  //**停止DSP執行程式動作
  run dsp 0                 //**啟動節點＃0 DSP執行程式動作
  run dsp 1                 //**啟動節點＃1 DSP執行程式動作
  prog exp1_pwm.plc         //**執行.PLC 批次檔
  das start                 //**啟動資料記錄
  das stop                  //**停止資料記錄
```

5.prog download.plc 執行節點模組 FPGA 下載批次檔

```
fpga piDSPv0.bit 0;
fpga piDSPv0.bit 1;
NET 1 1;
NET;
run dsp 0;
run reset;
```

fpga ＜name> <n> //**下載各節點 FPGA 檔案格式

fpga piDSPv0.bit 0;是指將 piDSPv0.bit DSP核心程式下載至節點＃0

fpga piDSPv0.bit 1;是指將 piDSPv0.bit DSP核心程式下載至節點＃1

NET 1 1; NET； 是指網路節點設定＃0至＃1

實驗程序設計

批次檔範例：exp1_pwm.plc

```
echo off;                                   //**畫面顯示開始
run dsp 0;                                  //**選擇節點DSP#0
run stop; run reset;                        //**既有程式停止
run "testDCM.mem";                          //**下載指定程式
echo off;                                   //**畫面顯示開始
echo clear;
echo "************************************";
echo "*  實驗一: 開環路的電壓控制          *";
echo "************************************";
echo "CH1：控制命令     CH2：馬達模擬";
echo "CH3：馬達速度     CH4：馬達轉角";
```

```
echo " ";
echo "模組使用狀況：";
echo "IO1:PHT位置速度輸入界面      ";
echo "FG1:方波產生器               ";
echo "    =>FT1:馬達模擬用濾波器   ";
echo "    =>IO2: PWM電壓輸出界面   ";
echo display;                               //**畫面顯示結束

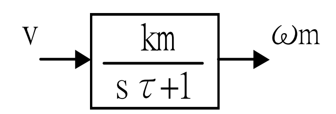
das 1 &.fg1.out 0 1024 "PWM 波形" "Volt";  //**設定DAS頻道#1
das 2 &.ft1.lp  0 1024 "馬達模擬" "Volt";  //**設定DAS頻道#2
das 3 &.io1.spd 0 1024 "馬達速度" "Volt";  //**設定DAS頻道#3
das 4 &.io1.ang 0 1024 "馬達角度" "Volt";  //**設定DAS頻道#4
plot select 0;                              //**設定示波器頁數
plot xunit 100000 0;                        //**設定示波器X軸
plot yunit 1 500   -300;                    //**設定示波器Y1軸
plot yunit 2 500   -100;                    //**設定示波器Y2軸
plot yunit 3 50     100;                    //**設定示波器Y3軸
plot yunit 4 50000  300;                    //**設定示波器Y4軸
plot trigger 1 0 1 10;                      //**設定示波器觸發
plot scope 9 1 2 3 4;                       //**設定示波器顯示

module .io2;                                //**設定io2模組
s cmd &.fg1.out;
module .fg1;                                //**設定fg1模組
s kAmp 256; s kFrq 150;
module .ft1;                                //**設定ft1模組
s u   &.fg1.out; s dt 10; s ka 500;
module .io3;                                //**設定io3模組
s led &.fg1.out;

module .;                                   //**回復預設模組
run 1khz;                                   //**設定抽樣頻率為1KHz
run start;                                  //**DSP開始執行
das stop; pause 1 "DAS啟動"; das start;     //**DAS啟動
echo on;                                  //**回復正常訊息顯示
```

直流伺服馬達開迴路的電壓控制

實驗說明：以電壓直接控制轉速的開迴路特性，可簡化為一階系統處理，方塊圖如下：



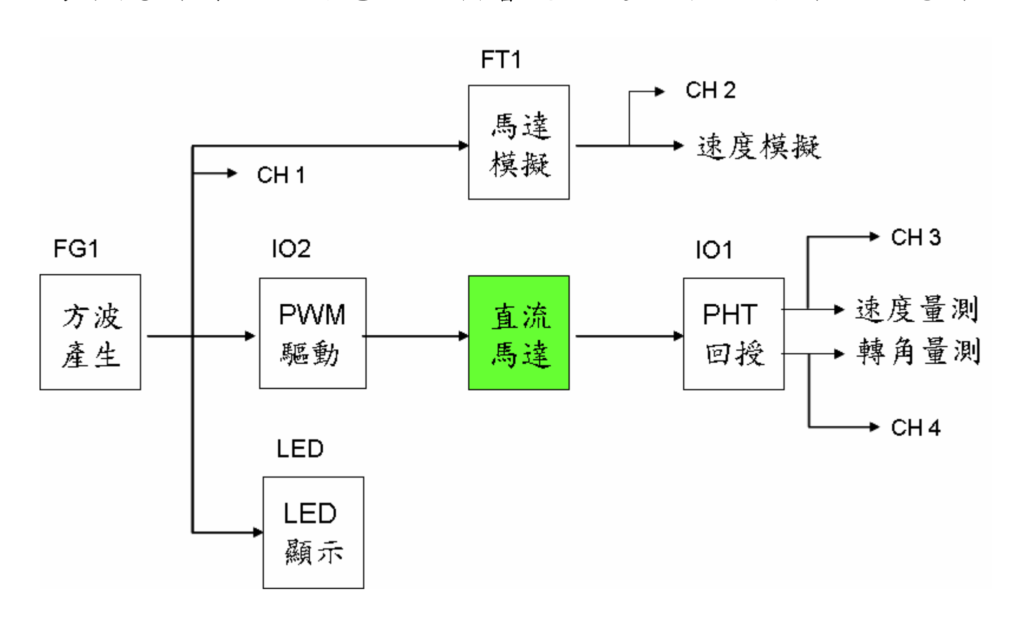
其中，V ：送給驅動器的PWM電壓命令。

ωm：馬達的轉動速度，可由處理photo encoder信號所得的位置微分而得。

km：一階系統的穩態增益。

τ ：一階系統的時間常數，即步階響應中的上升時間。

為了觀測開迴路的電壓控制響應，建立下面的測試方塊圖：



其中：

1. 用fg1產生測試用的方波訊號透過io2模組將訊號送出給伺服介面。
2. 用ft1建立對照用的一階系統，可用IIR1濾波器方塊產生。
3. 將方波命令產生PWM電壓，再觀測PHT(io1)上的速度值spd。
4. 調整IIR1濾波器的參數，使得兩組波形完全相符。這時IIR1濾波器就是電壓控制的開環路特性。其中km為增益，而τ為上升時間(單位為sec)。
5. 增益km可從示波器中作簡單計算，而上升時間(τ=4096/4000*dt/ka)的表格列舉如下。

IIR1參數dt	10	5	2	1	1	1	1
IIR1參數ka	256	256	256	256	512	1024	2048
上升時間τ	0.04	0.02	0.008	0.004	0.002	0.001	0.0005

6. 記錄km和τ，可進行系統的參數估測並進行後續的補償器設計工作。

7.可用LED顯示模組顯示命令數字。

其中的設計重點包括：

1.這是一個實驗範例，其他所有的實驗程序都有著類似的結構。
2.以標準的echo指令結構顯示本章的操作畫面，包括顯示畫面、清除按鍵和重新定義按鍵等動作。

```
echo off;                                          //**畫面顯示開始
echo clear;
echo "*************************************";
echo "*  實驗一: 開環路的電壓控制          *";
echo "*************************************";
echo "CH1：控制命令     CH2：馬達模擬";
echo "CH3：馬達速度     CH4：馬達轉角";
echo " ";
echo "模組使用狀況：";
echo "I01:PHT位置速度輸入界面     ";
echo "FG1:方波產生器              ";
echo "   =>FT1:馬達模擬用濾波器   ";
echo "   =>I02: PWM電壓輸出界面   ";
echo display;                                      //**畫面顯示結束
```

實際顯示畫面如下：



文字功能說明檔，配合 exp1_pwm.plc 設計內容

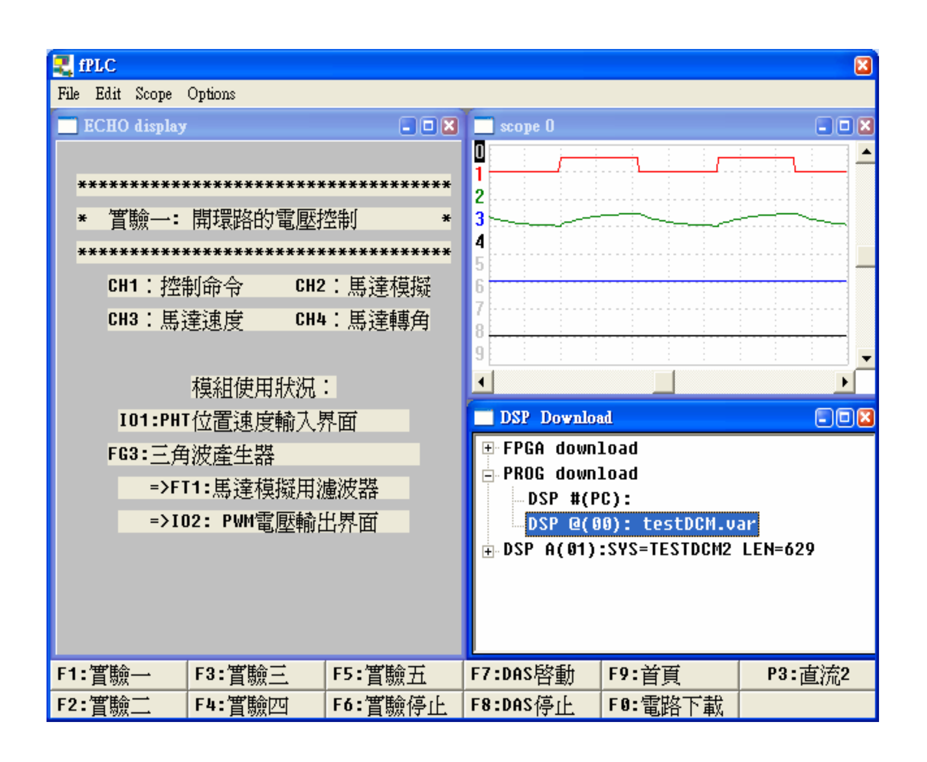
3. 載入系統程式，run "testDCM.mem";

testDCM.mem 是 testDCM.sys 經過 eLINK 連結程式產生之檔案。

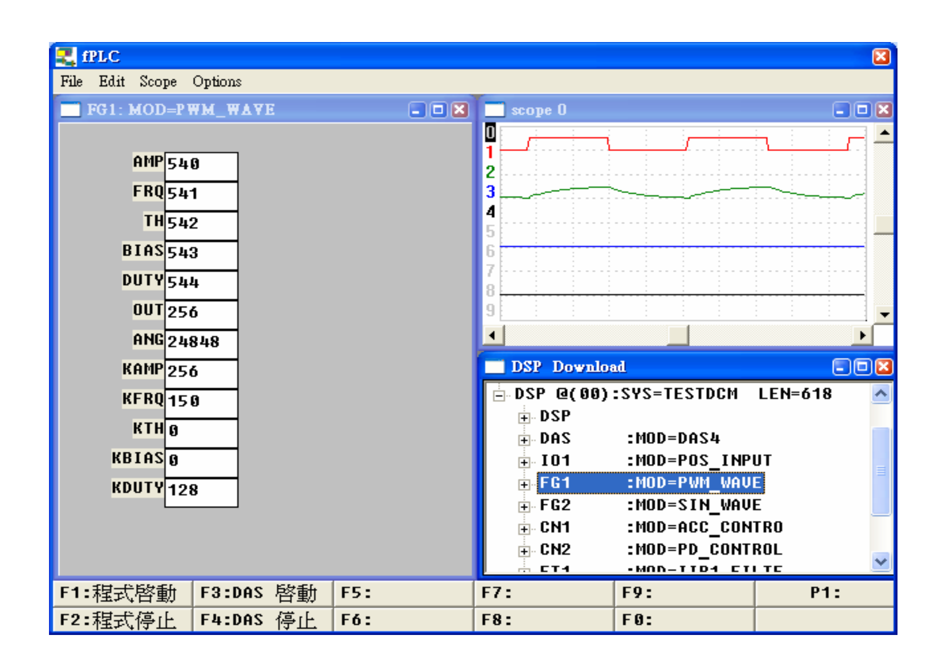
系統程式範例：testDCM.sys

```
@module   das   das4.mod;          //示波器資料蒐集
@module   io1   POS_input.mod;     //位置和速度讀值
@module   fg1   PWM_wave.mod;      //方波產生器
@module   fg2   SIN_wave.mod;      //弦波產生器
@module   cn1   ACC_control.mod;   //加減速控制器
@module   cn2   PD_control.mod;    //比例微分控制器
@module   ft1   IIR1_filter.mod;   //一階濾波器
@module   io2   PWM_output.mod;    //PWM電壓輸出界面
@module   io3   LEDKEY.mod;        //LED/KEY界面
```

上面所有宣告之模組為直流伺服馬達控制實驗，共 5 項實驗所使用之控制程式，其中 das4.mod; //示波器資料蒐集控制模組程式，使用 piDSP 組合語言撰寫，再經過 eASM 編譯後產生 *.mod 檔，為了下面參數宣告方便，以 das 變數宣告。



當載入系統程式，run "testDCM.mem";時 DSP @(00) var 顯示該檔案



將 DSP A（00）：前面 [+] 以滑鼠點選時，此時右下角會將 testDCM.sys 所有宣告模組顯示，只要再點選任何模組，再按 DSP Download ，左邊會將該模組所有參數即時顯示出來。

```
das 1 &.fg1.out 0 1024 "PWM 波形" "Volt";  //**設定DAS頻道#1
das 2 &.ft1.lp  0 1024 "馬達模擬" "Volt";  //**設定DAS頻道#2
das 3 &.io1.spd 0 1024 "馬達速度" "Volt";  //**設定DAS頻道#3
das 4 &.io1.ang 0 1024 "馬達角度" "Volt";  //**設定DAS頻道#4
plot select 0;                             //**設定示波器頁數
plot xunit 100000 0;                       //**設定示波器X軸
plot yunit 1 500   -300;                   //**設定示波器Y1軸
plot yunit 2 500   -100;                   //**設定示波器Y2軸
plot yunit 3 50     100;                   //**設定示波器Y3軸
plot yunit 4 50000  300;                   //**設定示波器Y4軸
plot trigger 1 0 1 10;                     //**設定示波器觸發
plot scope 9 1 2 3 4;                      //**設定示波器顯示
```

das 1 &.fg1.out 0 1024 "PWM 波形" "Volt"; //**設定 DAS 頻道#1
將方波產生器模組（fg1）輸出（out）宣告在 DAS 頻道#1，以"PWM 波形"為該名稱，並以 0 至1024 "Volt" 為單位，此時表示並無任何意義。

das 2 &.ft1.lp 0 1024 "馬達模擬" "Volt"; //**設定 DAS 頻道#2
馬達模擬模組（ft1）輸出埠（lp）宣告在 DAS 頻道#2

das 3 &.io1.spd 0 1024 "馬達速度" "Volt"; //**設定 DAS 頻道#3
光電盤的回授模組（io1），馬達速度的量測值（spd）宣告在 DAS 頻道#3

das 4 &.io1.ang 0 1024 "馬達角度" "Volt"; //**設定 DAS 頻道#4
光電盤的回授模組（io1），馬達角度的量測值（ang）宣告在 DAS 頻道#4

plot select 0; //**設定示波器頁數
設定示波器頁數，以 select 0 至 9 選擇 SCOPE 0 至 9 。

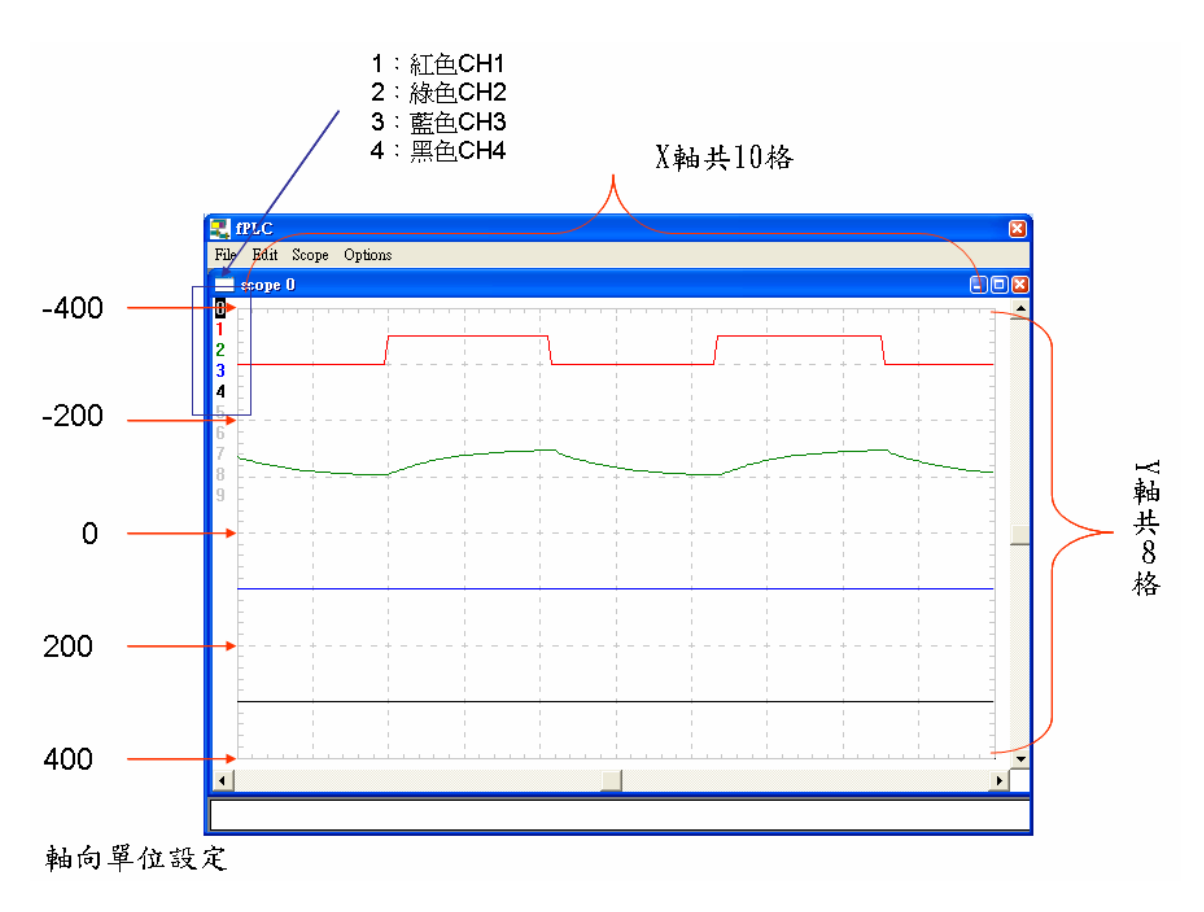
plot yunit 1 500 -300; //**設定示波器Y1軸
示波器第1頻道Y軸設定為：每格單位為 500，在 -300 位置

plot yunit 2 500 -100; //**設定示波器Y2軸
示波器第2一頻道Y軸設定為：每格單位為500，在 -100 位置

plot yunit 3 50 100; //**設定示波器Y3軸
示波器第3頻道Y軸設定為：每格單位為50，在 100 位置

plot yunit 4 50000 300; //**設定示波器Y4軸
示波器第4頻道Y軸設定為：每格單位為50000，在 300 位置

下圖為軸向設定說明：



plot xunit 100000 0;

X 軸向單位設定：

在示波器功能 100000 表示以一格單位 us 來計算，$100000*10^{-6}$ 每格為 0.1秒，走完 10 格需要 1 秒，從 0 為起始點。（示波器最快量測信號為 32 KHz）

在邏輯分析功能 100000 表示以一格單位 ns 來計算，$100000*10^{-9}$ 。（邏輯分析功能最快量測信號為 24 MHz）

plot trigger 1 0 1 10; //**設定示波器觸發

觸發信號設定，共4種功能

1 ：觸發模式設定。

0 ：觸發 Slope 設定，可選定為 0 為正向觸發，1 為負向觸發。

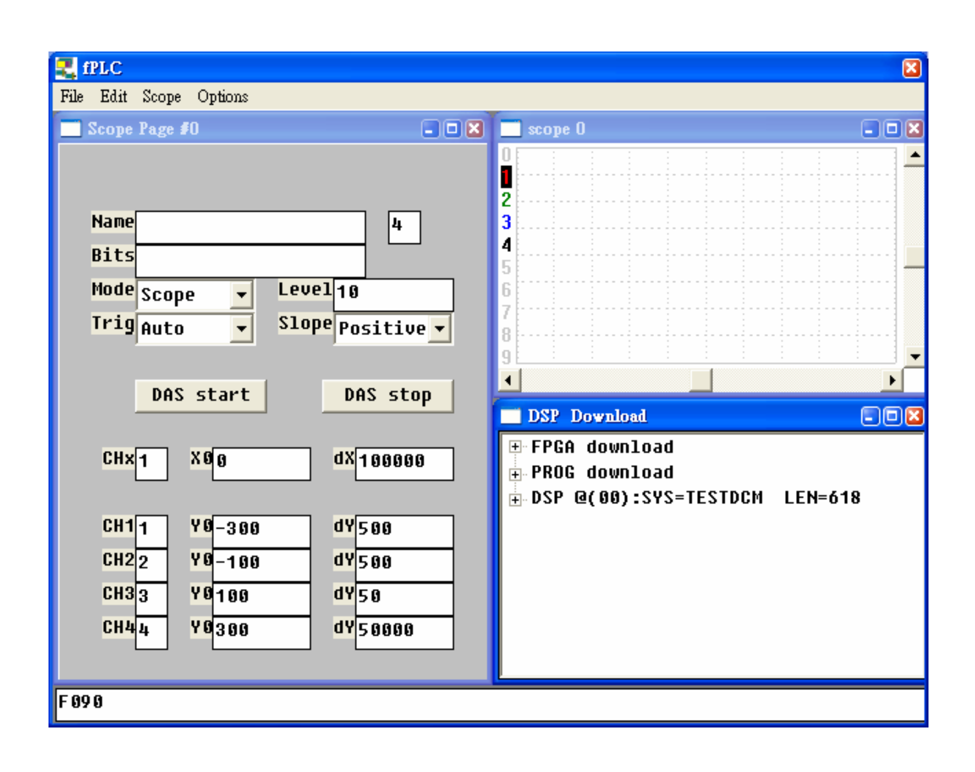
1 ：觸發頻道選擇，以 1 至 4 選擇。

10 ：觸發準位值，以範例一，以方波為準位 0 至 256 為命令波形，只要取該範圍內即可。

plot scope 9 1 2 3 4; //**設定示波器顯示

宣告以 9 示波器模式， 並以 1 2 3 4 ，4頻道功能顯示。

以上功能可在點選 scope 0 時就可顯示該設定功能，並同時也可更改其內容，並改變其狀況。



根據實驗一直流伺服馬達開迴路的電壓控制控制方塊圖，模組連線設定如下：

1. 所使用的模組包括：
   - FG1：方波產生器，產生所需的步階測試波形。
   - FT1：馬達模擬，以一階系統的低通濾波器來執行。
   - IO1：光電盤的回授處理，可同時取得轉角和速度的量測值。
   - IO2：PWM電壓驅動，直接驅動直流馬達作控制。
   - Led：四位數字七段顯示模組，顯示波形信號資料。

```
module .io2;                                      //**設定io2模組
s cmd &.fg1.out;
module .fg1;                                      //**設定fg1模組
s kAmp 256; s kFrq 150;
module .ft1;                                      //**設定ft1模組
s u   &.fg1.out; s dt 10; s ka 500;
module .io3;                                      //**設定io3模組
s led &.fg1.out;
```

```
module .io2;                                      //**設定io2模組
s cmd &.fg1.out;
```

將方波產生器（fg1）輸出（out）連結至PWM電壓驅動輸入埠（cmd）

```
module .fg1;                                      //**設定fg1模組
s kAmp 256; s kFrq 150;
```

設定方波產生器（fg1）內部參數初始值，振幅（kAmp）與頻率（kFrq）。

```
module .ft1;                                      //**設定ft1模組
s u   &.fg1.out; s dt 10; s ka 500;
```

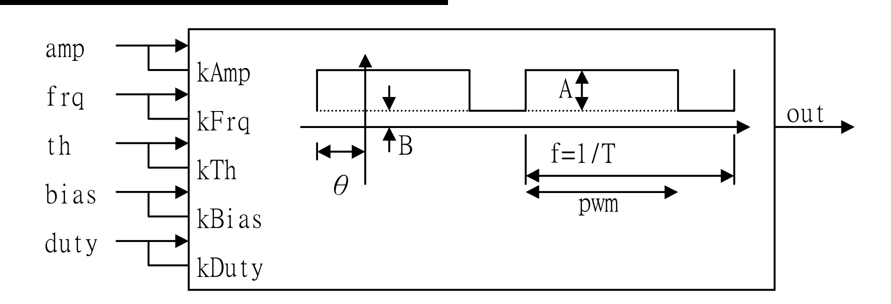
設定馬達模擬（ft1）內部參數初始值，ka 與 dt。

```
module .io3;                                      //**設定io3模組
s led &.fg1.out;
```

將方波產生器（fg1）輸出（out）連結至四位數字七段顯示模組輸入埠（led）

以下為這 5 個模組組合語言說明檔：

脈衝波產生器



此模組會產生週期性的脈衝波。方塊圖如下：

其中：

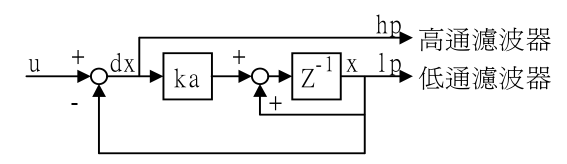
1. 脈衝波有五個參數，其中振幅(A)、頻率(F)、相位(θ)和偏移(B)四個參數和正弦波定義完全一樣，另外再增加一個負載率(D)。
2. 基本波形是從週期開始時為1，隔段時間後換為0，然後持續到週期結束再重新開始。其中1所佔有的比例就是負載率，當D=0時為0%，而D=255時為100%。
3. 和其他參數類似，負載率(D)以duty為其輸入點，而以kDuty為其參數值。

模組程式範例：PLS_WAVE.asm

```
@local  amp   R0  IP kAmp;          //輸入點，脈衝波的振幅
@local  frq   R1  IP kFrq;          //輸入點，脈衝波的頻率
@local  th    R2  IP kTh;           //輸入點，脈衝波的相角
@local  bias  R3  IP kBias;         //輸入點，脈衝波的中心偏移
@local  duty  R4  IP kDuty;         //輸入點，脈衝波的負載率
@local  out   R5  OP;               //輸出點，脈衝波輸出
@local  ang   R6;                //內部變數，相位角，65536=360°
@local  kAmp  R7  1000;             //參數，振幅值
@local  kFrq  R8  1000;          //參數，頻率值=65536/8000*Hz
@local  kTh   R9  0;               //參數，相角值，360°=256
@local  kBias R10 0;               //參數，偏移值
@local  kDuty R11 128;             //參數，負載率，100% =256
@data   12;                        //模組變數區長度及初值
@initial:                             //啟動程序
     WR ang,#0; WR out,#0;            //ang=out=0
     RET;
@timer:                             //即時控制程序(8KHz)
     RD  ang;    ADD (frq); WR ang;            //ang+=frq
     SFR 8;      ADD (th);  AND #255;        //ACC=(ang/256+th)&0x00ff
     SUB (duty); RD  (amp); JLT t1; RD #0;     //ACC=(amp,0)
t1:  ADD (bias); WR   out;                     //out=ACC+bias
     RET;
```

IIR一階濾波器

此模組執行一階的IIR濾波器，方塊圖如下：



濾波器參數的計算公式如下：

1. 假設濾波器的頻寬為hz，則
   ω_n=hz*6.28
2. 若系統頻率為8KHz，則抽樣週期dT為
   dT=dt/8000
   其中dt為模組參數。
3. 則濾波器參數為：
   ka=ω_n*dT*4096，或是
   =hz*dt*6.28/8000*4096，即
   =hz*dt*3.2
4. 當dt決定後，所允許的訊號頻寬為(4000/dt)。換句話說，訊號頻寬會因為dt增加而下降。

為了方便實驗的運用，下面列出常用的參數對照表：

濾波頻寬 (Hz)	10	20	50	100	200	500	1000	2000
訊號頻寬 (Hz)	400	800	2000	4000	4000	4000	4000	4000
模組參數dt	10	5	2	1	1	1	1	1
模組參數ka	320	320	320	320	640	1600	3200	6400

表格的運用方式如下：

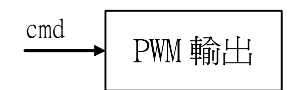
1. 首先決定濾波頻寬。
2. 接著查表找出對映的參數dt和ka。
3. 再查表得到對映的訊號頻寬，以決定是否需要串接前級濾波器。

模組程式範例：IIR1_FILTER.asm

```
@local  u    R0 IP;          //輸入點，濾波器輸入
@local  lp   R1 OP;          //輸出點，低通濾波器輸出
@local  hp   R2 OP;          //輸出點，高通濾波器輸出
@local  dt   R3 1;           //參數，抽樣週期
@local  ka   R4 4096;        //參數，增益ka，1.0=4096
@local  prd  R5;             //內部變數，抽樣週期用
@data   6;                   //模組變數區長度
@initial:           //***** 開機啟動程序 *****
     WR lp,#0; WR hp,#0; WR prd,#1; RET;      //lp=hp=0,prd=1
@timer:             //***** 即時控制程序 *****
     SUB prd,#1; JNZ b9;            //if (--prd>0) return
     RD dt;  WR prd;                              //prd=dt
     RD (u); SUB lp; WR hp;                       //hp=u-lp
     MUL ka; SFR 12; ADD lp; WR lp;        //lp+=(hp*ka/4096)
b9:  RET;
```

PWM 輸出控制模組

模組方塊：



程式說明：

1.在開機時將切換保護時間寫到輸出阜0xbc上，其中切換保護值為0~63，60=5us。
2.在每次timer interrupt時將cmd訊號處理後寫到輸出阜0xb8上。

PWM_output.asm程式列印：

```
@local  cmd    R0   IP gnd;        //輸入點，v相命令
@local  gain   R1   4096;          //參數，輸出增益，1.=4096
@local  bias   R2   2048;          //參數，輸出偏壓，0.=2048
@local  limit  R3   4095;          //參數，輸出極限，1.=4095
@local  zone   R4   0x88f0;  //參數，頻率=12KHz,切換保護=5us
@local  pwm    R5   0xb8;    //輸出阜地址，PWM#0 輸出(0xb8)
@local  mode   R6   0xbc;    //輸出阜地址，PWM   模式(0xbc)
@local  p0md   R7   0xf8;      //輸出阜地址，port#0模式(0xf8)
@local  gnd    R8   0;                  //輸入接地點
@data   9;                              //模組變數區長度
```

```
@initial:              //***** 開機啟動程序 *****
     RD  bias;   WR (pwm);           //pwm=bias
     RD #0xffff; WR (p0md);          //set port#0 mode
     RD  zone;   WR (mode); RET;     //mode=zone
@stop:                  //***** 關機保護程序 *****
     RD  bias;   WR (pwm);  RET;     //pwm=bias
@timer:                //***** 即時控制程序 *****
     RD (mode); xor #1; and #1; WR (mode); //trigger watch-dog
     RD (cmd); MUL gain;  SFR 12; ADD bias;//ACC=(cmd*gain/4096)+bias
     MAX #0;   MIN limit; WR (pwm);         //pwm=limit(ACC)
     RET;
```

光電盤入模組

DC 馬達的位置，由光電盤量測之，預設儲存於 0xac 與 0xbe 二個輸入埠位址中。其中位址 0xac 記錄 X 軸馬達的位置，位址 0xbe 記錄Y軸馬達的位置。由於本實驗的 DC 馬達接在Y軸的位置上，因此程式中，直接選用 Y 軸的輸入。

光電盤位置輸入的程式列印如下，其中速度由位置的差分而得。至於馬達的位置，則使用內部 2 個 word 加以記錄，精度為 32 位元。

POS_input.asm程式列印：

```
@local  spd    R0 OP;                //輸出點，速度量測值
@local  ang    R1 OP;                //輸出點，角度量測值
@local  angH   R2;                   //內部變數，角度的32位元部分
@local  mode   R3 0x308;          //模式選擇，1.5MHz,A/B相，正/反向輸入
@local  pht    R4 0xac;                   //輸入阜地址，光電盤輸入(0xac)
@local  phtc   R5 0xbe;                   //輸入阜地址，光電盤模式(0xbe)
@local  old    R6;                        //內部變數，速度更新用
@data   7;                                //模組變數區長度
@initial:                                 //***** 開機啓動程序 *****
     RD mode;  WR (phtc);                   //phtc=mode
     RD (pht); WR old; WR spd,#0; RET;      //old=pht,spd=0
@timer:                                   //***** 即時控制程序 *****
     RD (pht); SUB.X old; SGN 11; WR spd;   //spd=(pht-old),old=pht
               ADD ang;   WR ang;           //angHL+=spd
     RD.H;     ADC angH;  WR angH;
     RET;
```

四位數字七段顯示模組

```
//*********************************************************
//** LEDKEY.asm : 測試LED和KEYBOARD的響應
//**
//** xPA0~3：LED數字掃瞄，   對映到P2[0~3]
//** xPC0~3：鍵盤輸入點，    對映到P2[4~7]
//** xPB0~7：七段式LED顯示，對映到P3[0~7]
//*********************************************************
@local  led R0  IP gnd;       //**輸入點，指向LED要顯示的資料
@local  key R1  OP;           //**輸出點，4*4按鍵的對映值
@local  cnt R2;               //**計時控制
@local  buf R3;               //**移位控制
@local  dat R4;               //**按鍵暫存
@local  scn R5;               //**掃瞄控制
@local  op2 R6  0x92;         //**輸出阜地址，GPIO#2輸出端
@local  op3 R7  0x93;         //**輸出阜地址，GPIO#3輸出端
@local  ip2 R8  0x9A;         //**輸出阜地址，GPIO#2輸入端
@local  ip3 R9  0x9B;         //**輸出阜地址，GPIO#3輸入端
@local  tab R10 TP;           //**segment[8]表格區的起始位址
@local  gnd R11 0;            //**輸入接地點
@data   12;                   //***** 模組變數區長度 *****
@table  tab 16;               //***** seg7[16]表格區 *****
    { 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07 }; //0~7
    { 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71 }; //8~F
@initial:                     //***** 開機啟動程序 *****
    RD #0; WR cnt; WR buf;                  //** cnt=buf=0
    RD #0xe;       WR scn;                  //** scn=0x000f
    RD #0xffff; WR (op2); WR (op3); RET;    //** op2=op3=0xffff
@stop:                        //***** 關機結束程序 *****
    RD #0xffff; WR (op2); WR (op3); RET;    //** op2=op3=0xffff
@timer:                       //***** 即時控制程序 *****
    ADD cnt,#1;                             //** cnt=cnt+1
    RD  cnt;  AND #3;     JNZ t9;           //** every 4 interrupt
    RD (op3); AND #0xff00; WR (op3);        //** SEG7=0
    RD scn,1; AND #0xf;    WR scn;          //** scn=scn<<1
    SUB #0xe; JNC t1; ADD scn,#1;       //** scn+=1 when scn!=1110
t1: RD (ip2); COM; SFR 4; AND #0xf;         //** AX=ip2[7~4]
    ADD dat,4;             WR dat;          //** dat=(dat<<4)+AX
    RD  scn;  AND #0xf;    WR.H;            //** HX=scn&0xf;
    RD (op2); AND #0xfff0; OR.H; WR (op2);//** op2=(op2&0xfff0)|HX
    RD  cnt;  AND #0xc;    JNZ t2;          //** for 4-digit
    RD  dat;       WR key;                  //** key=din;
    RD (led);      WR buf; JR t3;     //** buf=*led   when cnt=0
t2: RD buf; SFR 4; WR buf;            //** buf=buf>>4 when cnt>0
```

```
t3: AND #0xf; ADD tab; WR.H; RD.M; OR (op3);//** G7=tab[buf&0xf]
    WR (op3);
    t9: RET;
```

新實驗建立程序

綜合上述的這些步驟，一個新實驗的建立程序如下：

1. 首先確認所需的模組程式是否已經完全具備。如果是既有程式則可以直接拿來運用，如果必須自建新模組，則必須依照下述程序一一進行：
   A. 以文書編輯程式或 TaxPad 建立 ASM 檔的組合語言程式。
   B. 利用 eASM 程式將 ASM 檔編譯成模組程式(MOD 檔)。
2. 以文書編輯程式或 TaxPad 建立系統程式(SYS 檔)，再利用 eLINK 程式將之編譯成機械碼(MEM 檔)和變數檔(VAR 檔)。
3. 根據實驗控制程序，建立所需要的實驗程序(PLC 檔)。

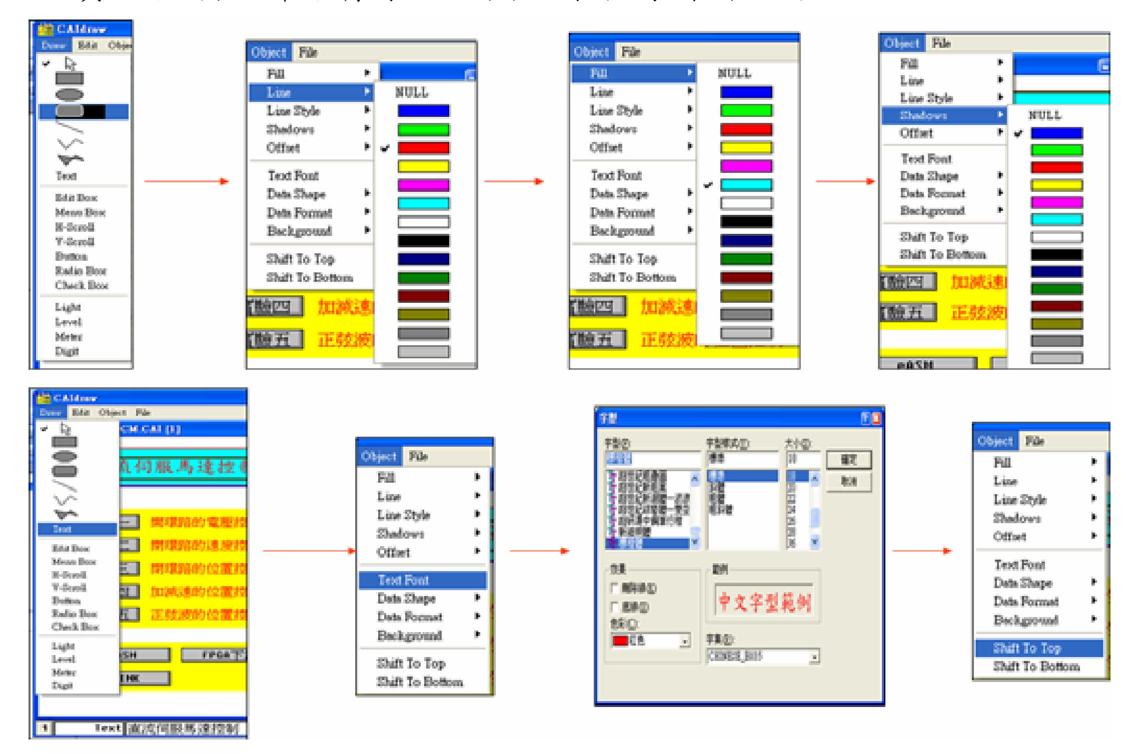
在所有程式準備完成後，即可依標準開機程序執行 exp_DCM.plc 程式。進入系統首頁後，再以功能按鍵的操作步驟選取適當實驗而執行。實驗執行後，如果發現程式錯誤或是參數設定值不妥，在找出對應程式做適當的修改。

2.3 CAI 指令說明

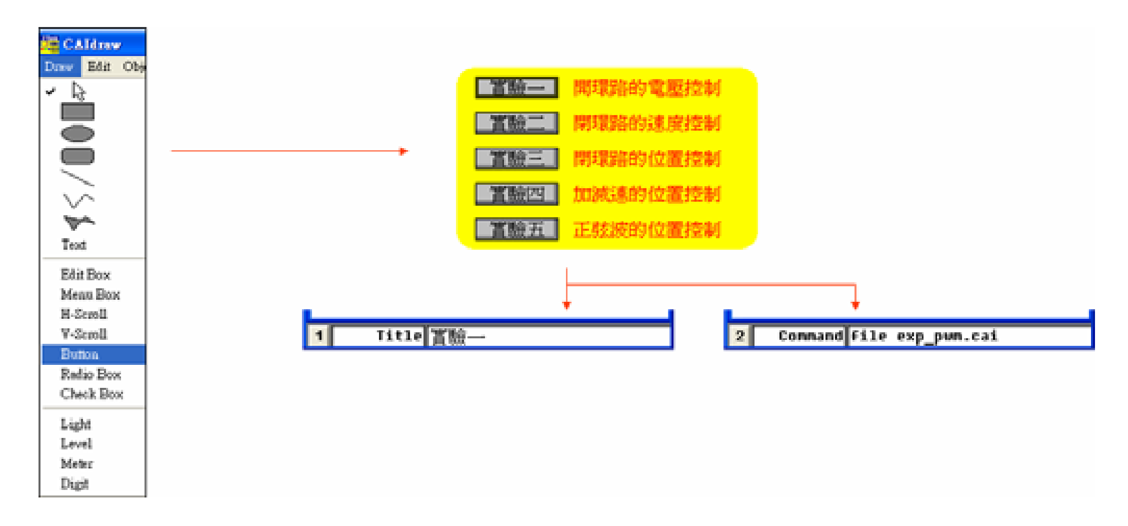
以直流伺服馬達控制實驗之首頁與實驗一達範例說明如下：
testDCM.cai



#1　實驗抬頭設計分背景框框與文字程序操作如下：



＃2　實驗項目選擇設計操作如下：



使用 Button 指令，以滑鼠拉引按鈕框框，在 Title 輸入：實驗一，再以鍵盤↓，出現 Command 輸入：連結實驗名稱 file exp_pwm.cai
，以此類推分別建立：
實驗二：file speed.cai
實驗三：file angle.cai
實驗四：file acc.cai
實驗五：file sin.cai

＃3　實驗項目說明設計操作如下：
操作方式如同＃1 步驟分別設計即可。

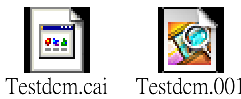
＃4　系統功能按鈕設計操作如下：
操作方式如同＃2 步驟分別設計即可，詳細內容如下：

eASM 按鈕 Command 內容：exec eASM %*.asm （執行編譯程式）

eLink 按鈕 Command 內容：exec eLINK %*.sys （執行系統連結程式）

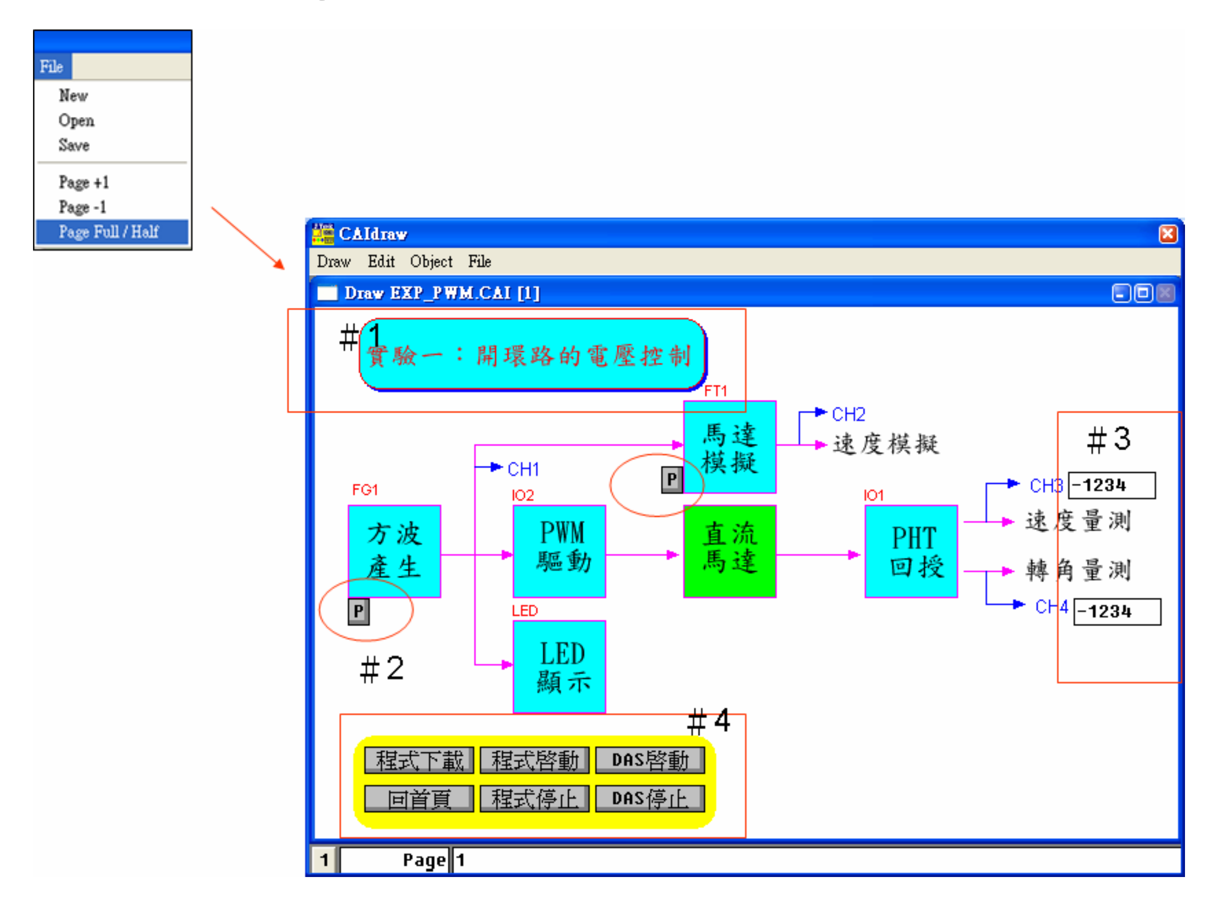
FPGA 下載按鈕 Command 內容：fpga piDSPv0.bit 0;run dsp 0;run reset
（將 piDSPv0.bit 檔案下載至 eM_USB ＃0 模組）

當您儲存檔案時會產生下面兩個檔案：





exp_pwm.cai　page 1 說明：



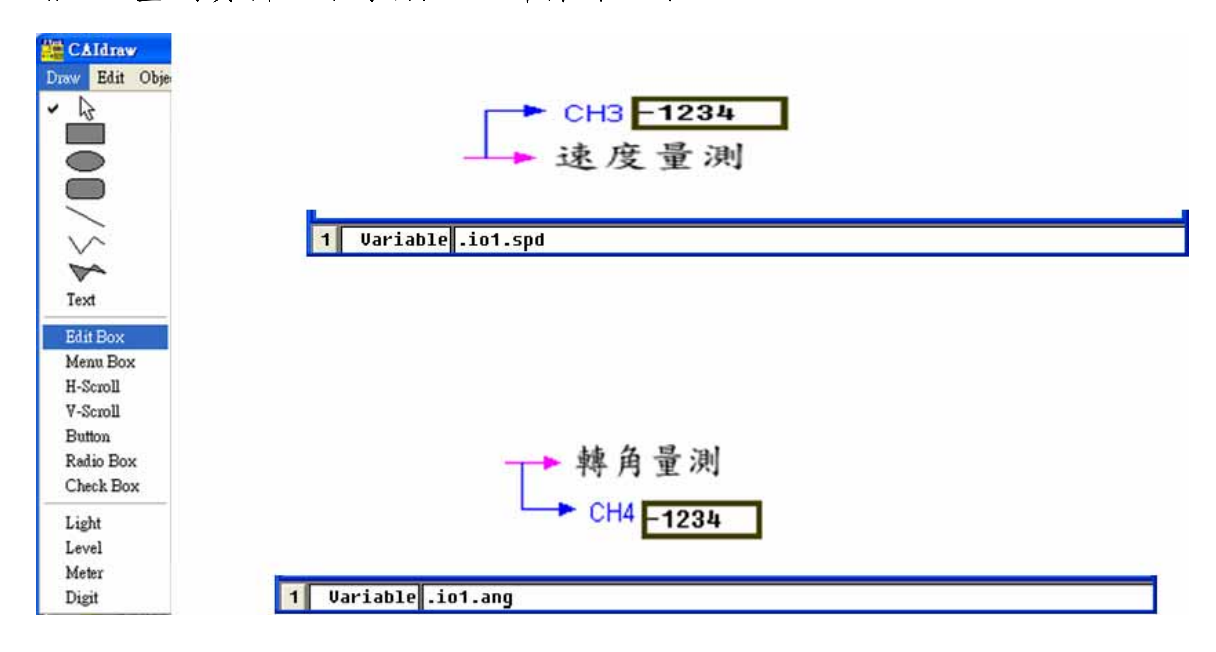
＃1　實驗抬頭設計分背景框框與文字程序操作如下：
操作方式如同 testDCM.cai ＃1 步驟方式設計即可。

＃2　參數功能按鈕設計操作如下：
操作方式如同 testDCM.cai＃2 步驟分別設計即可，詳細內容如下：

p 按鈕 Command 內容：page 2（跳到第二頁改變方波產生設定變數）

p 按鈕 Command 內容：page 3（跳到第三頁改變馬達模擬設定變數）

＃3　量測資料以數字顯示設計操作如下：



當建立 Edit Box 動態資料變數物件時，會要求輸入變數值，io1.ang 與 io1.spd 為 testdcm.sys 模組連結程式宣告 @module fg1 POS_input.mod （//方波產生器）組合語言程式變數，詳細組合語言程式可參考 2.2 PLC 實驗規劃說明、記住前面必須加 .（. io1.ang）。

```
@module   fg1   PWM_wave.mod;       //方波產生器
@module   fg2   SIN_wave.mod;       //弦波產生器
@module   cn1   ACC_control.mod;    //加減速控制器
@module   cn2   PD_control.mod;     //比例微分控制器
@module   io1   POS_input.mod;      //位置和速度讀值
@module   io2   PWM_output.mod;     //PWM電壓輸出界面
@module   ft1   IIR1_filter.mod;    //一階濾波器
@module   das   das4.mod;           //示波器資料蒐集
@module   io3   LEDKEY.mod;         //LED/KEY界面
```

＃4　系統功能按鈕設計操作如下：
操作方式如同 testDCM.cai＃2 步驟分別設計即可，詳細內容如下：

程式下載 按鈕 Command 內容：PROG exp_pwm.plc;PAGE 0 （執行實驗一 exp_pwm.plc 程式連結與設定）

回首頁 按鈕 Command 內容：file TESTDCM.CAI （回到實驗首頁）

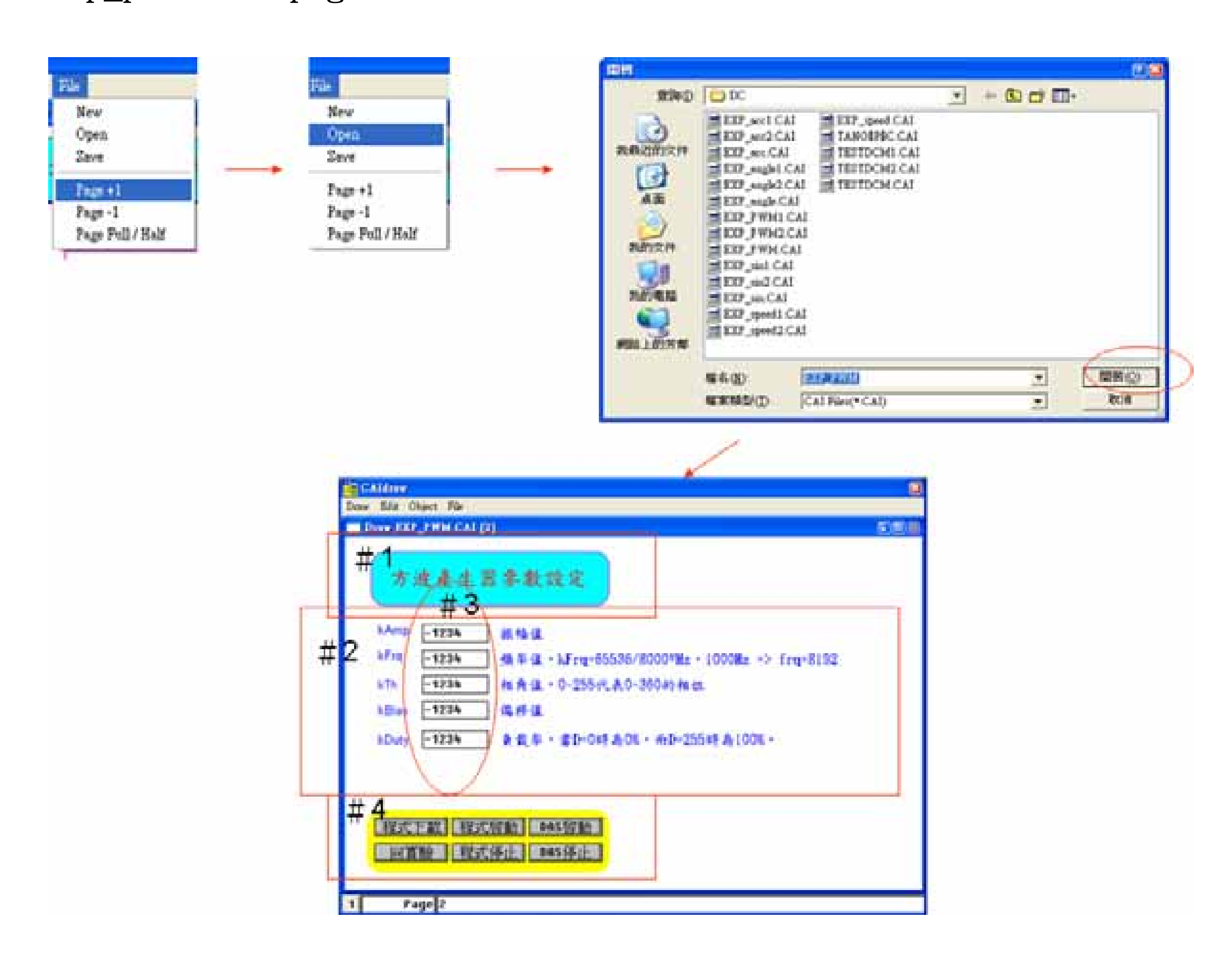
程式啟動 按鈕 Command 內容：RUN start（執行控制程式）

程式停止 按鈕 Command 內容：RUN stop（停止控制程式）

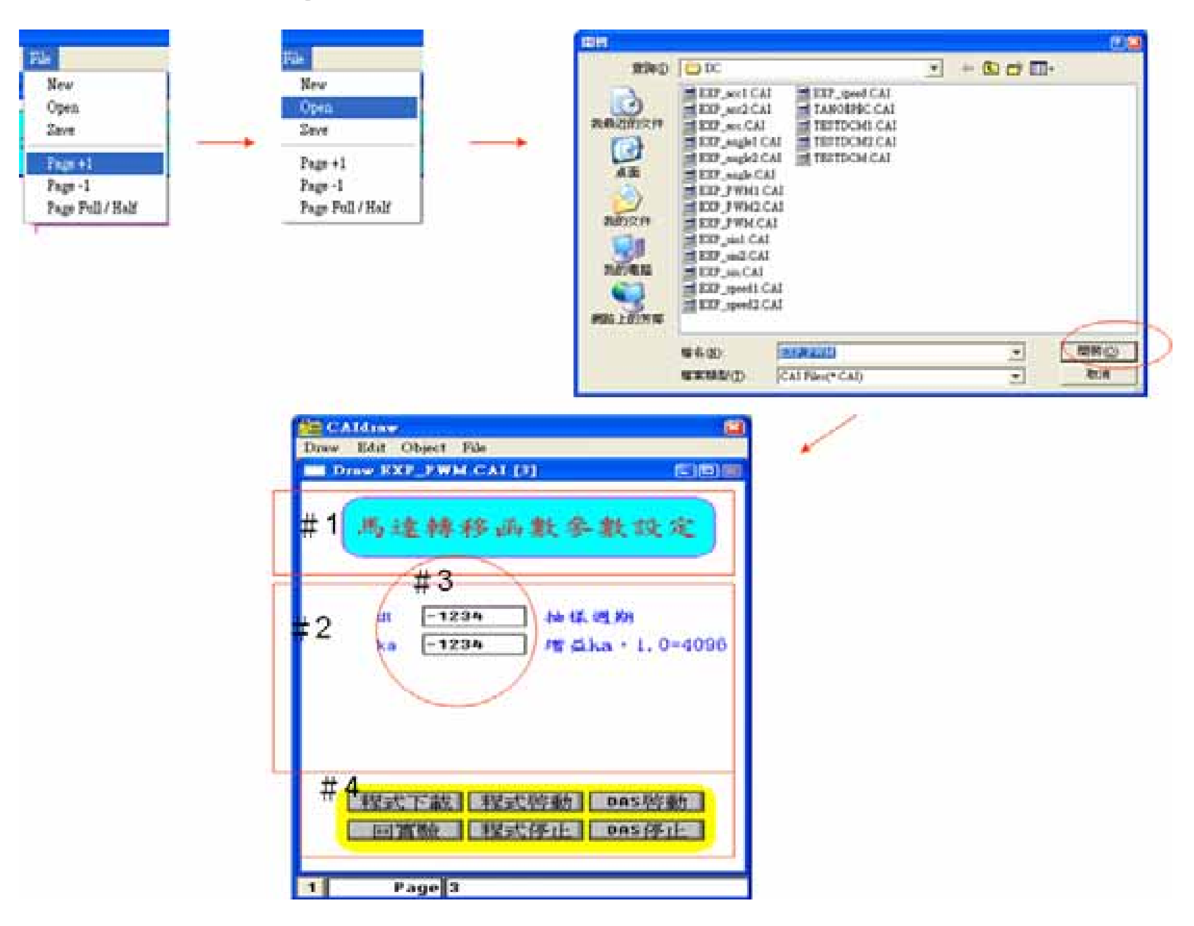
DAS啟動 按鈕 Command 內容：DAS start（執行資料記錄）

DAS停止 按鈕 Command 內容：DAS stop（停止資料記錄）

exp_pwm.cai　page 2 說明：



exp_pwm.cai　page 3 說明：



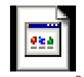
＃1　實驗抬頭設計分背景框框與文字程序操作如下：
操作方式如同 testDCM.cai ＃1 步驟方式設計即可。

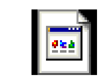
＃2　參數文字設計操作如下：
操作方式如同 testDCM.cai ＃1 步驟方式設計即可。

＃3　控制參數設定資料顯示設計操作如下：
當建立 Edit Box 動態資料變數物件時，會要求輸入變數值，ft1.dt 與 ft1.ka 為 testdcm.sys 模組連結程式宣告 @module　ft1 IIR1_fillter.mod; （//一階濾波器）組合語言程式變數，詳細組合語言程式可參考 2.2 PLC 實驗規劃說明。

＃4　系統功能按鈕設計操作如下：
操作方式如同 testDCM.cai 第二頁 ＃2 步驟分別設計即可。

當您儲存檔案時會產生下面四個檔案：


Exp_pwm.003


Exp_pwm.cai


Exp_pwm.001


Exp_pwm.002

最後連接 exp_pwm.plc 批次檔

批次檔範例：exp_pwm.plc

```
echo off;                                          //**畫面顯示開始
run dsp 0;                                         //**選擇節點DSP#0
run stop; run reset;                               //**既有程式停止
run "testDCM.mem";                                 //**下載指定程式

das 1 &.fg1.out 0 1024 "PWM 波形" "Volt";   //**設定DAS頻道#1
das 2 &.ft1.lp  0 1024 "馬達模擬" "Volt";   //**設定DAS頻道#2
das 3 &.io1.spd 0 1024 "馬達速度" "Volt";   //**設定DAS頻道#3
das 4 &.io1.ang 0 1024 "馬達角度" "Volt";   //**設定DAS頻道#4
plot select 0;                              //**設定示波器頁數
plot xunit 100000 0;                        //**設定示波器X軸
plot yunit 1 500   -300;                    //**設定示波器Y1軸
plot yunit 2 500   -100;                    //**設定示波器Y2軸
plot yunit 3 50     100;                    //**設定示波器Y3軸
plot yunit 4 50000  300;                    //**設定示波器Y4軸
plot trigger 1 0 1 10;                      //**設定示波器觸發
plot scope 9 1 2 3 4;                       //**設定示波器顯示

module .io2;                                //**設定io2模組
s cmd &.fg1.out;
module .fg1;                                //**設定fg1模組
s kAmp 256; s kFrq 150;
module .ft1;                                //**設定ft1模組
s u   &.fg1.out; s dt 10; s ka 500;
module .io3;                                //**設定io3模組
s led &.fg1.out;

module .;                                   //**回復預設模組
run 1khz;                                   //**設定抽樣頻率為1KHz
run start;                                  //**DSP開始執行
das stop; pause 1 "DAS啟動"; das start;     //**DAS啟動
echo on;                                  //**回復正常訊息顯示
```

以上檔案與 2.2 PLC 實驗規劃中 exp1_pwm.plc 最大差別為，實驗文字說明以 echo 顯示，全部刪掉，再以 CAI 圖控畫圖程式設計。

新實驗建立程序

綜合上述的這些步驟，一個新實驗的建立程序如下：

1. 首先確認所需的模組程式是否已經完全具備。如果是既有程式則可以直接拿來運用，如果必須自建新模組，則必須依照下述程序一一進行：
   A. 以文書編輯程式或 TaxPad 建立 ASM 檔的組合語言程式。
   B. 利用 eASM 程式將 ASM 檔編譯成模組程式(MOD 檔)。
2. 以文書編輯程式或 TaxPad 建立系統程式(SYS 檔)，再利用 eLINK 程式將之編譯成機械碼(MEM 檔)和變數檔(VAR 檔)。
3. 根據實驗控制程序，建立所需要的實驗程序(PLC 檔)。
4. 使用 CAI 圖控畫圖程式規劃設計人機操控畫面（CAI）。

在所有程式準備完成後，即可依標準開機程序執行 exp_DCM.cai 程式。進入系統首頁後，再以功能按鍵的操作步驟選取適當實驗而執行。實驗執行後，如果發現程式錯誤或是參數設定值不妥，在找出對應程式做適當的修改。

3. *CAIdraw* 圖控程式發展工具

CAIdraw 之特色與優點

一個好的控制系統發展軟體除了強調即時控制軟體的功能外，對於「人機介面」的處理也是非常重要的一環，尤其在Windows作業環境下的圖形操作方式，如何設計出一個具有親和力的操作畫面，對於系統整體來說是不可或缺的功能。

市面上有許多控制系統，雖然在硬體及控制程式上有不錯的設計，但是在人機介面程式上的處理卻往往被忽略掉，甚至根本不提供此方面的支援，須由使用者自己去想辦法(如使用Visual Basic、Visual C++等)，或者另外再買一套昂貴的「圖控程式發展系統」來搭配，增加了使用者在金錢上、時間上許多額外的負擔。

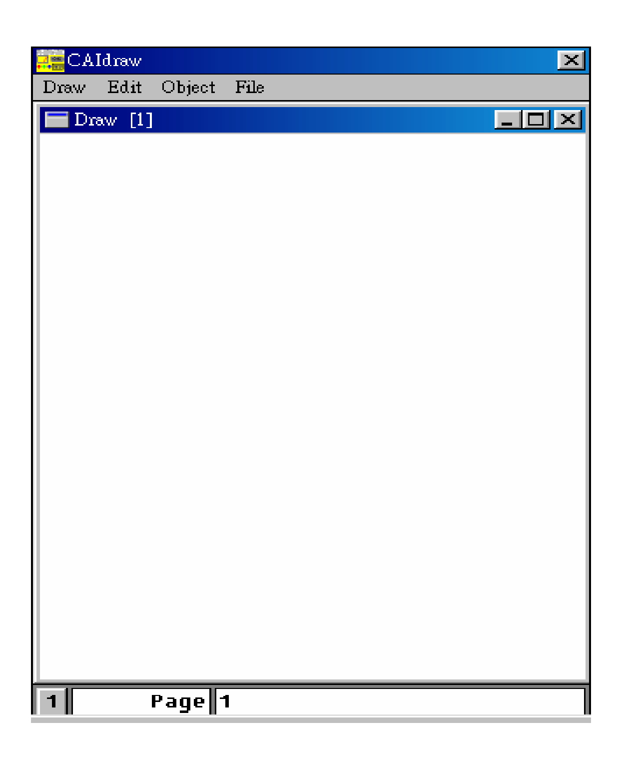
在fPLC這套系統中，就完全解決了此方面的問題，『*fPLC* 系統群組』內有一個可以用來撰寫『圖控程式』的應用程式—CAIdraw，其不僅具有好學易用的特色，更具備了Windows環境下常見的輸出/輸入元件及一般圖控軟體所提供的控制元件，使你輕輕鬆鬆地設計出極具親和力的操作畫面來，而且不需要你再花任何經費。

3.1 啟動 *CAIdraw*

當你安裝好RFFPGA 檔案後後，您只要在TESTASM 子目錄下單擊 

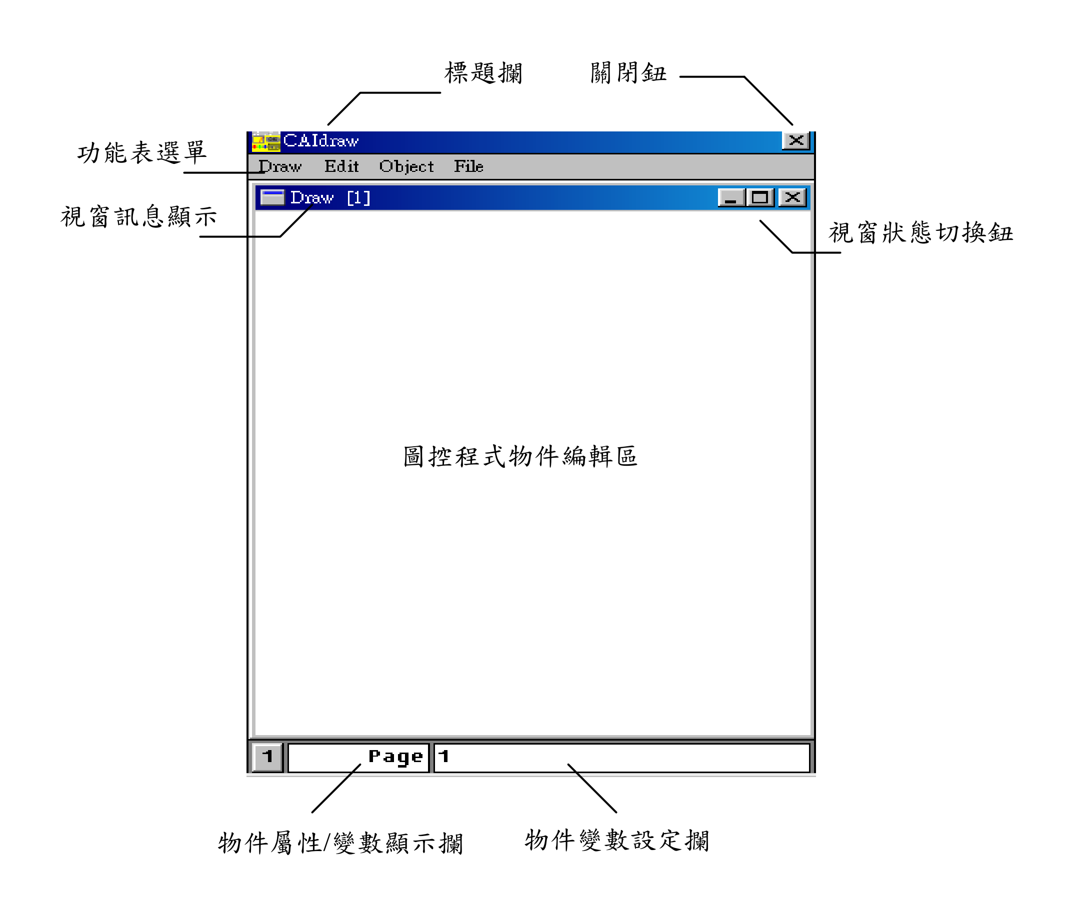
CAI_draw.exe

會出現 CAIdraw 視窗，如下圖所示：



3.2 認識 *CAIdraw* 視窗環境

當你進入CAIdraw視窗後，就可以開始編輯、撰寫你的圖控程式。基本上，CAIdraw視窗的外型與一般Windows的應用程式視窗相似，其視窗結構圖可以下圖來說明：



上面所標示各欄位之用途如下：

標題欄

顯示CAIdraw應用程式的名稱，若有多個視窗同時被打開時，可由此標題欄之顏色判斷是否為正在使用的視窗(顏色較深)。

關閉鈕

按下此鈕可立即關閉CAIdraw應用程式視窗，而且不會顯示任何警告訊息，所以在沒有存檔之前千萬不要按下此鈕，否則所修改的內容將不被接收。

功能表選單

此功能表包含CAIdraw應用程式所擁有的功能，包含『Draw』、『Edit』、『Object』、『File』等四大群組，各群組均具有若干子功能選項，經由滑鼠點選各功能表選單後，會出現下拉式的功能表單供使用者選擇。詳細之使用方式將在第四節中說明。

視窗訊息顯示

此訊息表示目前所選擇的CAIdraw工作視窗模式。本系統具有兩種視窗模式，可由『Edit』功能項的『1 Draw』或『2 Edit』來切換，系統內定為"Draw"模式。

視窗狀態切換鈕

關閉視窗鈕 可將CAIdraw下的「Draw」視窗或「Edit」視窗關閉。

放到最大鈕 可將CAIdraw下的「Draw」視窗或「Edit」視窗放大至最大範圍。

縮到最小鈕 可將CAIdraw下的「Draw」視窗或「Edit」視窗暫時隱藏，可以用「放到最大鈕」或「還原鈕」來恢復視窗。

還原鈕 可將CAIdraw下的「Draw」視窗或「Edit」視窗

恢復至正常的操作狀況。

(注意！一般情況下，以上各功能鈕沒有切換的必要，盡量不要用)

圖控程式物件編輯區

此區域即繪製圖控程式各物件的工作區，使用者即在此區域內設定、繪製其圖控程式物件。

物件屬性/變數顯示欄

顯示現在所選擇物件的類別屬性(靜態圖形類、動態變數資料類)及其相關的組成變數名稱。當你以滑鼠點選「箭頭指標」切換成「選擇物件模式」後，就可以用箭頭指標選擇任何物件(指到物件後按一下滑鼠左鍵)，此時「物件屬性/變數顯示欄」即會馬上顯示此物件之類別，若是屬於純粹靜態繪圖的物件(如：畫矩形、畫直線、文字標示等)則此欄位空白(沒有顯示任何訊息)，若是屬於動態的變數資料物件(如：「Edit Box」、「H-scroll」、「Meter」等)則顯示『Variable』字樣，除了在第一欄顯示物件的類別屬性外，並且可以「↑」、「↓」方向鍵來切換至各欄位，以便觀看或設定此物件之相關組成變數(如：「Minimum」、「Maximiml」、「Command」等)，詳細用法在功能表之「Draw」用法介紹時將一一說明。

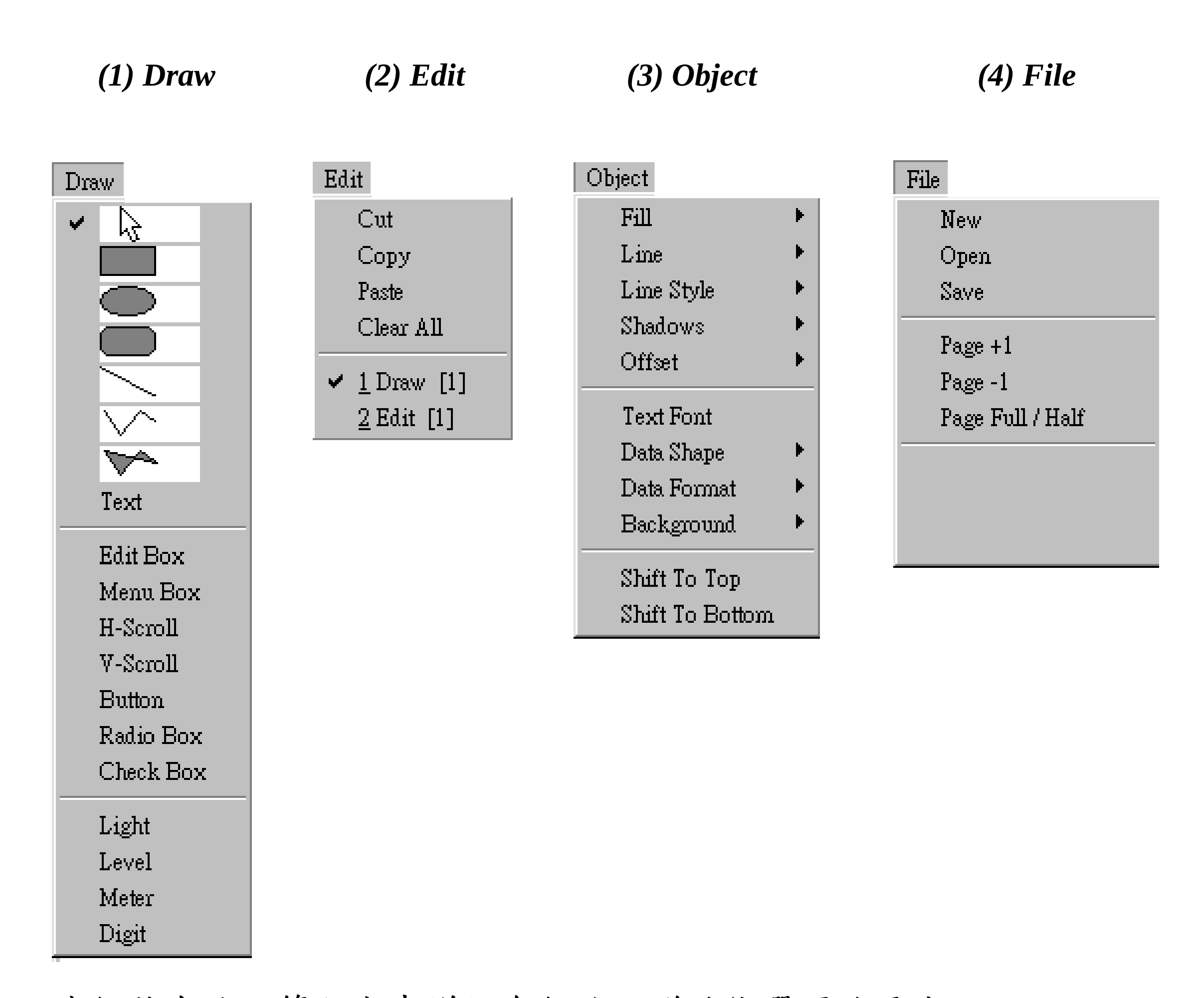
物件變數設定欄

此欄位乃配合前項「物件屬性/變數顯示欄」而設，當你選到動態的變數資料物件時，可使用「↑」、「↓」方向鍵來切換至各組成變數欄位，然後在各欄內填入你所需要的組成

變數數值，來設定物件在程式執行時的顯示狀況。詳細用法在功能表之「Draw」用法介紹時將一一說明。

3.3 *CAIdraw* 功能表介紹

CAIdraw視窗的「功能表選單」，包含『Draw』、『Edit』、『Object』、『File』等四大群組，各有其專屬功能，由其字面上的意義即可大約猜到負責哪一方面的工作，操作者只要以滑鼠左鍵在其上面點選一下，即會出現相關之下拉式子功能項，如下圖所示：

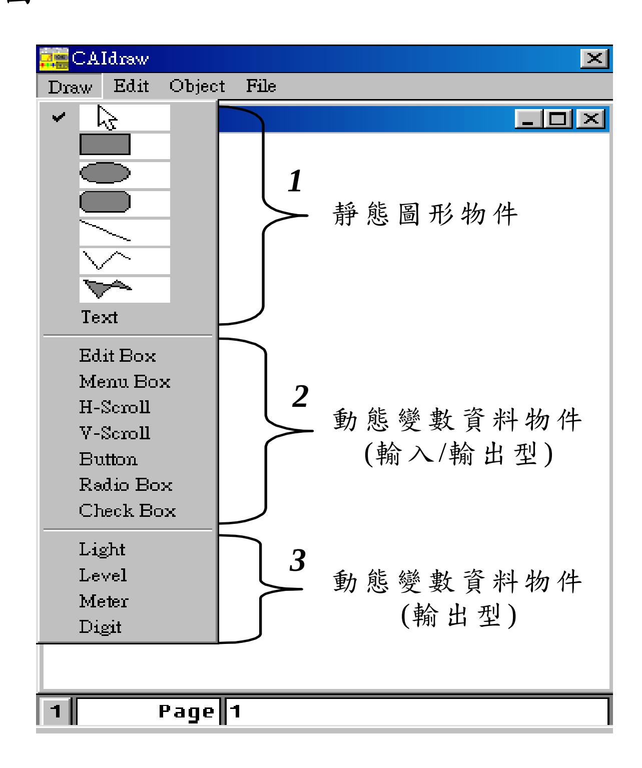


我們將在這一節內容中詳細介紹這四群功能選項的用法：

『Draw』—控制物件之繪製

在CAIdraw視窗中編寫圖控程式是一件非常輕鬆的事情，而其之所以輕鬆的原因，就在擁有「Draw」功能了。有了它，操作者不僅可以在工作區內利用簡單的繪圖工具繪製靜態的圖形物件，更可以使用各式系統內建的「動態變數資料物件」來設計各種「控制元件」(即各種動態之輸出、輸入圖控元件)，使得你所設計出的圖控程式畫面具有專業的水準，而且不必撰寫任何一行程式，只要選擇「Draw」功能下的各物件來〝繪製〞控制元件，然後填入相關之變數資料即可，就像小孩子塗鴉般簡單。不囉唆！馬上來介紹：

當你進入CAIdraw視窗後，以滑鼠左鍵點選一下「Draw」功能表單，就會出現以下畫面：



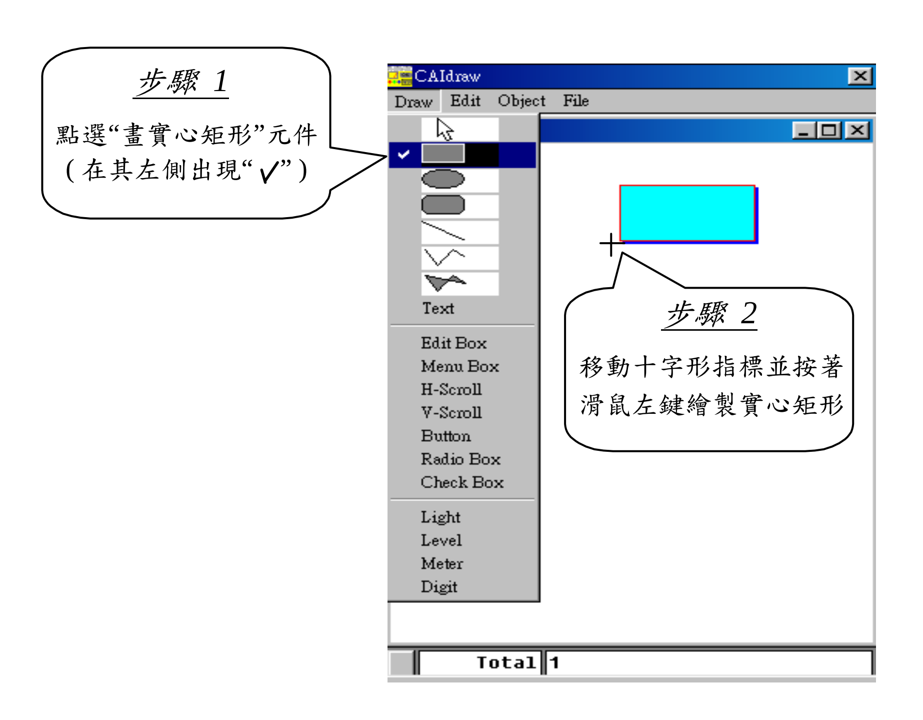
「Draw」功能表以橫隔線分成三區，如上圖所標示，其中：第1項屬於「靜態圖形物件」，與一般繪圖軟體之繪圖工具類似，可繪製矩形、橢圓形、直線、曲線、不規則多邊形、文字等，可搭配「Object」功能項來改變顏色、線條種類等外觀，對於增加畫面的美觀與生動有極大的幫助。

第2項屬於「動態變數資料物件」，可設計各種同時兼具輸入、輸出雙向溝通功能的控制元件，對於系統或程式的變數設定與顯示，具有非常靈活的表現方式，而且只要在fPLC視窗上執行時，就可以各種顯示模式即時顯示變動中的變數狀況，是非常典型的圖控元件。

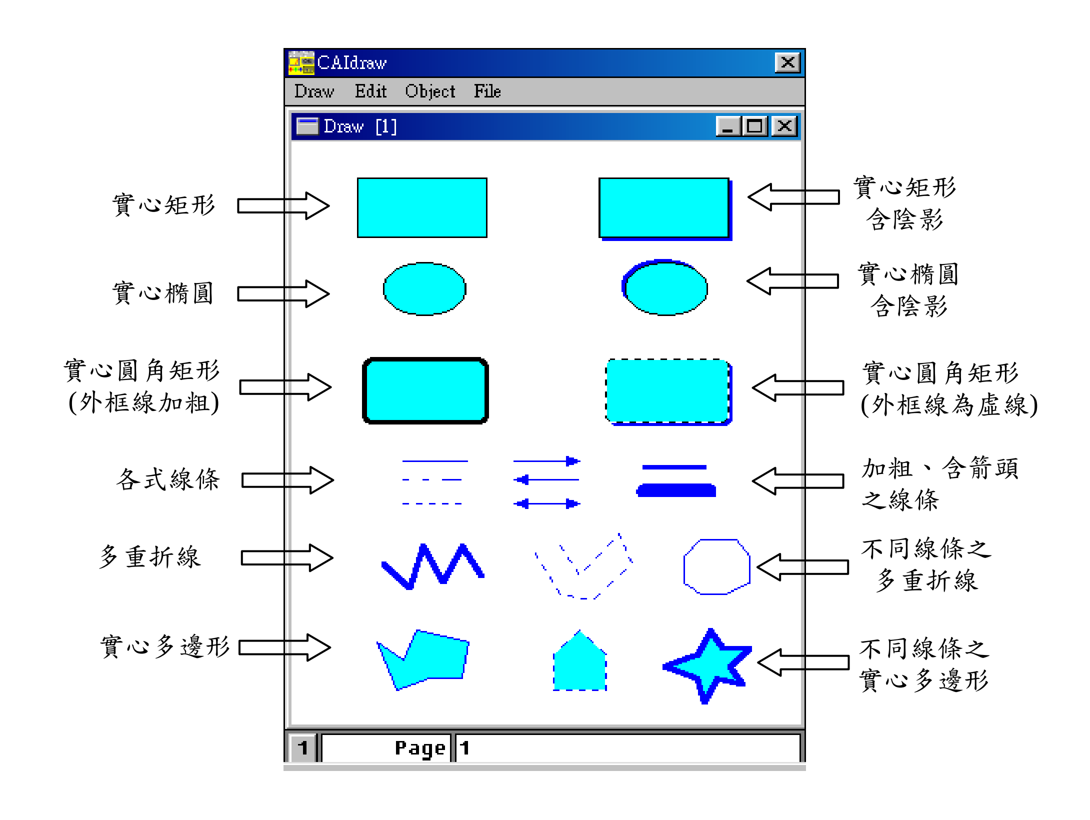
第3項也是屬於「動態變數資料物件」，不過只具有輸出功能 (即顯示功能)，無法作為輸入元件，但是本項元件在顏色及外型上的設定比第2項來得靈活，對於增加圖控畫面的生動效果，有極大的幫助。

以「靜態圖形物件」繪製圖形

當你點選「Draw」功能表後，請用滑鼠左鍵在子功能項(Pop-Up menu)的第1群中，點選你所要的元件。此時，滑鼠指標會由〝箭頭〞模式變成〝十字形〝模式，並且在所選的元件左側便會出現一個〝打勾〞的記號，提醒操作者現在所選的元件是哪一個。這樣你就可以移動此十字形指標到工作區內繪製你的元件了。操作步驟如下圖所示：



其他「靜態圖形物件」的使用方式，可以用相同的步驟來畫出，下圖列出經常用到的繪圖物件所繪出的圖案，供您參考：



其中有一些圖形效果須搭配「Object」功能項的修飾功能(如：Fill、Line、Line Style、Shadows、Offset等)才可做到，詳細的說明，請看「Object」功能項的使用。

以「靜態圖形物件」編寫文字

除了線條、圖形的繪製外，「文字顯示」功能也是設計圖控畫面時不可或缺的，在「Draw」功能表第1群的最後一項—【Text】就是為此而設計。不過文字顯示的操作比一般繪圖元件多一點步驟，如下說明：

步驟1：

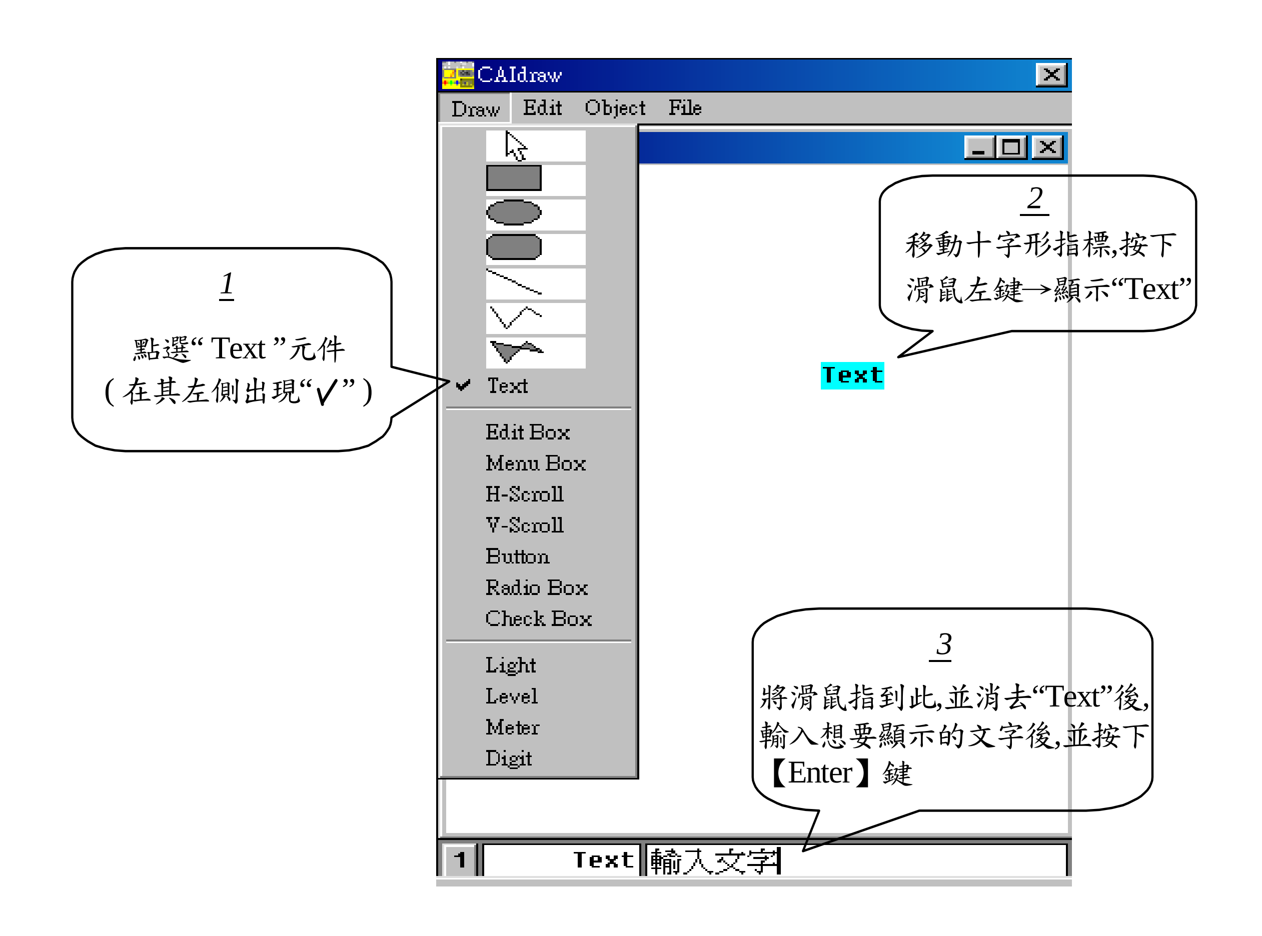
1.按【Draw】功能表

2.選【Text】功能

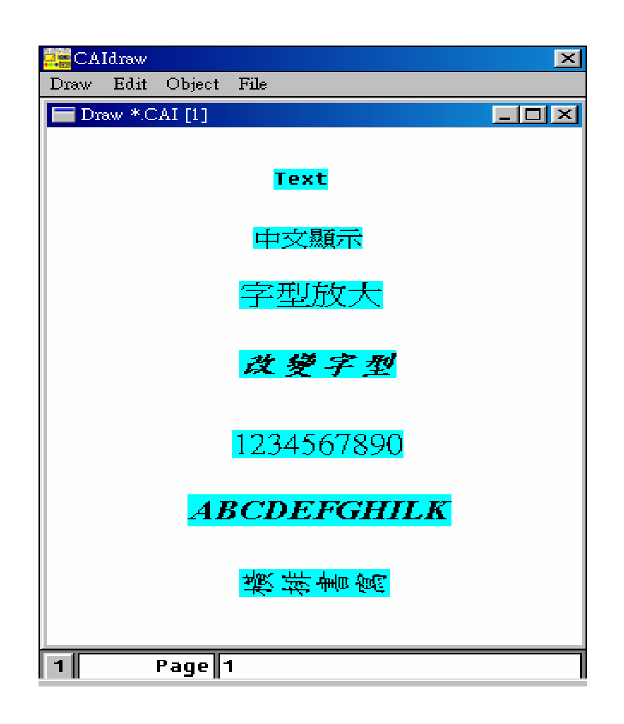
3.移動十字形指標至適當位置後按滑鼠左鍵一下→出現〝Text〞字樣。

步驟2：

1. 按【Draw】功能表
2. 選 】】功能 (“箭頭指標”表示目前為”選取物件”模式)
3. 指到剛才所設的〝Text〞處並按一下滑鼠左鍵→視窗底部出現”Text”字樣
4. 將滑鼠指到視窗最底下「物件變數設定欄」內的〝Text〞上，並按一下左鍵
5. 以〝←〞(back space 鍵)去掉〝Text〞文字，然後輸入想要顯示的文字
6. 按下【Enter】鍵即完成。如下圖所示：



如果搭配「Object」下的「Text Font」功能，可以設定更多的字形樣式，下圖列出一些常用到的字形設定：



以「動態變數資料物件」繪製控制元件

【Draw】功能表的第2、3群子功能項屬於變數控制元件類，每一個元件均需要對應到一個變數(Rn)，而且元件會隨著變數的更改而變動(動態元件)，或操作者也可藉著調整、設定元件而更改所對應的變數內容，所以與第1群元件只是單純的靜態圖形有很大的不同。事實上，「動態變數資料物件」處理的對象就是一些fPLC中系統或程式的變數(Rn)，將單調的數字式顯示或鍵盤key in的方式改成較具親和力的圖形式元件，而這些控制元件也經常在Windows軟體中出現，大家均非常熟悉，以下我們將一一為大家介紹：

Edit Box — **數字盒** (輸入/輸出型)

將變數以外型固定的數字形式顯示出來，並且具有一外框，長度可自行拖曳滑鼠左鍵來設定。內定形式為「十進位」，可藉由「Object」的【Data Format】來設定不同基底的數字系統(二進位、十六進位等)。由於屬於輸入/輸出型元件，所以除了顯示功能外，可藉由滑鼠在方框點選後，由鍵盤輸入數字後按【Enter】來執行變數設定(輸入)的功能。

步驟1：

1.按【Draw】功能表

2.選【Edit Box】功能 (指標變成十字形)

3.移動指標到工作區所要繪製的地方

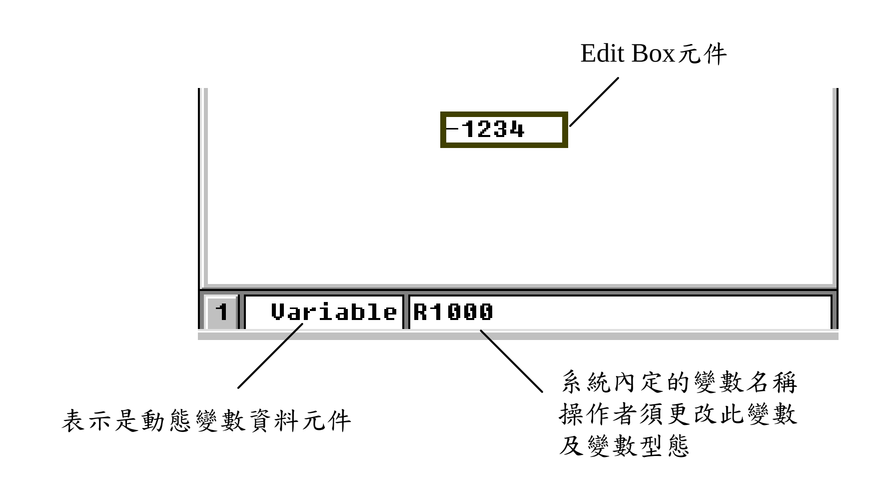
4.按下滑鼠左鍵以「拖曳」的方式繪製一長方形外框，然後釋放滑鼠左鍵，就會產生一個標示著〝-1234〞的數字方框

步驟2：

1.按【Draw】功能表

2.選【 】功能

3.指到剛才所繪的數字方框處，並按一下滑鼠左鍵→視窗底部兩欄位分別出現"Variable"、"R1000"字樣，前者表示此物件為"動態變數資料"元件，後者則是系統自動為此物件所指定的變數名稱及變數型態(R1000為1 Byte)，你必須為元件設定適當的變數名稱及型態以符合程式的的需要。如下圖所示：



4.將滑鼠指到視窗最底下「物件變數設定欄」內的〝R1000〞上，並按一下左鍵以〝←〞(back space鍵)去掉〝R1000〞文字，然後輸入適當的變數名稱及型態(如：R2000.W 註："W" 表示為"Word"型態之變數佔2 Bytes)

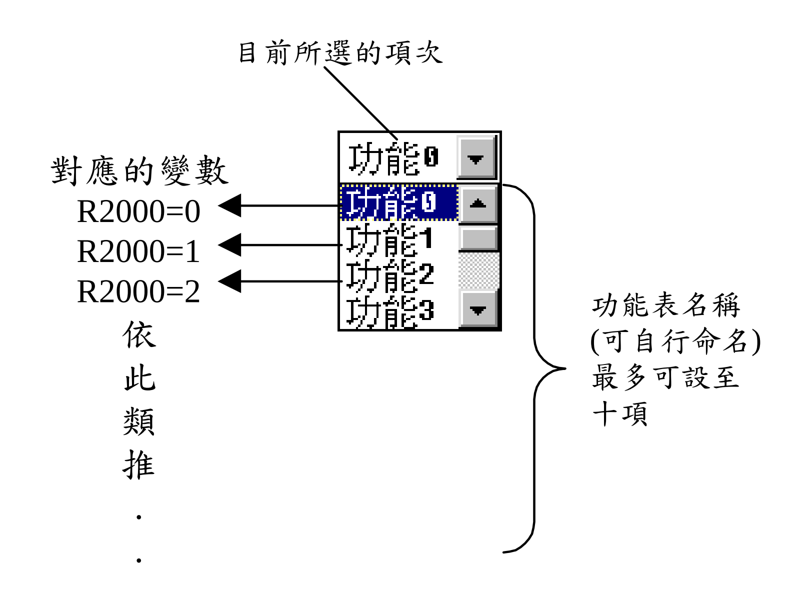
5.按下【Enter】鍵即完成設定。

其餘的「動態變數資料元件」均以類似方式來設定，只是不同的

元件具有不同的「元件組成變數」要設定（可能不只一個，藉由”↑”、”↓”方向鍵來切換）罷了。

Menu Box — **下拉式功能表單**（輸入/輸出型）

將變數以〝下拉式表單〞(Pop-Up Menu)的方式來表示，此類元件只能對應至Byte整數型變數(如：R2000,R100等)，而其數值之變化範圍也只是從0到9，其「元件組成變數」共有十項，從” Item0 ”至” Item9 ”，可由操作者自行輸入功能表名稱，而依其順序分別代表所設的變數值為0至9，其關係圖如下：



步驟1：

1.按【Draw】功能表

2.選【Menu Box】功能（指標變成十字形）

3.移動指標到工作區所要繪製的地方

4.按下滑鼠左鍵以「拖曳」的方式繪製一長方形外框，然後釋放滑鼠左鍵，就會產生一個「下拉式的功能表」

步驟2：

1.按【Draw】功能表

2.選【 】功能

3.指到剛才所繪的「下拉式功能表」處，並按一下滑鼠左鍵→視窗底部兩欄位分別出現”Variable”、”R1000”字樣。

4.將滑鼠指到視窗最底下「物件變數設定欄」內的〝R1000〞上，並按一下左鍵以〝←〞(back space鍵)去掉〝R1000〞文字，然後輸入適當的變數名稱及型態(如：R3000 表示為Byte型態之變數佔1 Byte)，並按下【Enter】鍵。

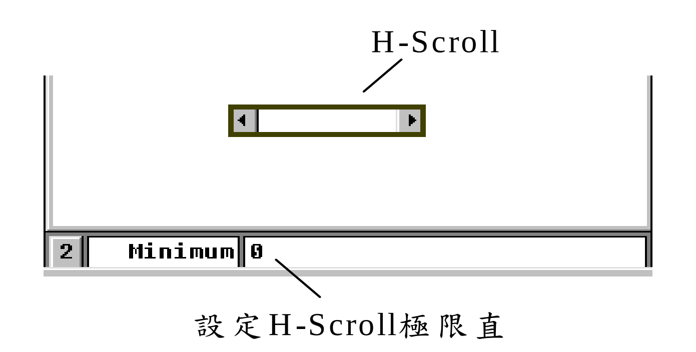
5.按一下〝↓〞，在視窗底部出現”Item0”字樣，將滑鼠指到「物件變數設定欄」內並按左鍵，出現輸入模式的閃爍直線。

6.鍵入第一個功能表選項的名稱（中英文皆可），並按【Enter】鍵。

7. 再按一下〝↓〞，出現”Item1”字樣，依此類推地完成各項功能表選項名稱的設定。

H-Scroll — **水平式捲軸** (輸入/輸出型)

將變數以〝水平式捲軸〞(Scroll bar)的方式來表示，可以用滑鼠按著捲軸兩邊的箭頭或拖曳中間的方框來更改變數值，而不須藉由鍵盤的key in，非常方便。此類元件適用於整數及浮點數（具小數點），具有兩組「元件組成變數」—『Minimum』、『Maximum』，分別用來設定捲軸的左極限（方框移至最左邊時）及右極限（方框移至最右邊時）之數值，內定值分別為0、100。其外型如下圖所示：



步驟1：

1.按【Draw】功能表

2.選【H-Scroll】功能 (指標變成十字形)

3.移動指標到工作區所要繪製的地方

4.按下滑鼠左鍵以「拖曳」的方式繪製一長方形外框，然後釋放滑鼠左鍵，就會產生一個「水平式捲軸」

步驟2：

1.按【Draw】功能表

2.選【 】功能

3.指到剛才所繪的「水平式捲軸」處，並按一下滑鼠左鍵→視窗底部兩欄位分別出現”Variable”、”R1000”字樣。

4.將滑鼠指到視窗最底下「物件變數設定欄」內的〝R1000〞上，並按一下左鍵以〝←〞(back space鍵)去掉〝R1000〞文字，然後輸入適當的變數名稱及型態(如：R2500.F 表示為Float型態之變數佔4 Byte)，並按下【Enter】鍵。

5.按一下〝↓〞，在視窗底部出現”Minimum”字樣，將滑鼠指到「物件變數設定欄」內並按左鍵，出現輸入模式的閃爍直線。

6.將內定值0刪去，再輸入左極限之數值，並按【Enter】鍵。

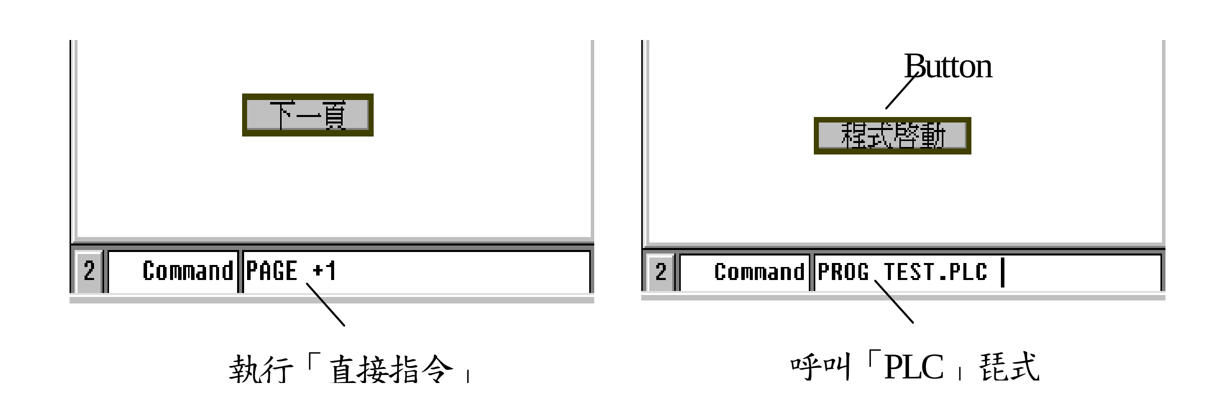
7.再按一下〝↓〞，出現"Maximum"字樣，將內定值100刪去，再輸入右極限之數值，並按【Enter】鍵，即完成設定。

V-Scroll — **垂直式捲軸**（輸入/輸出型）

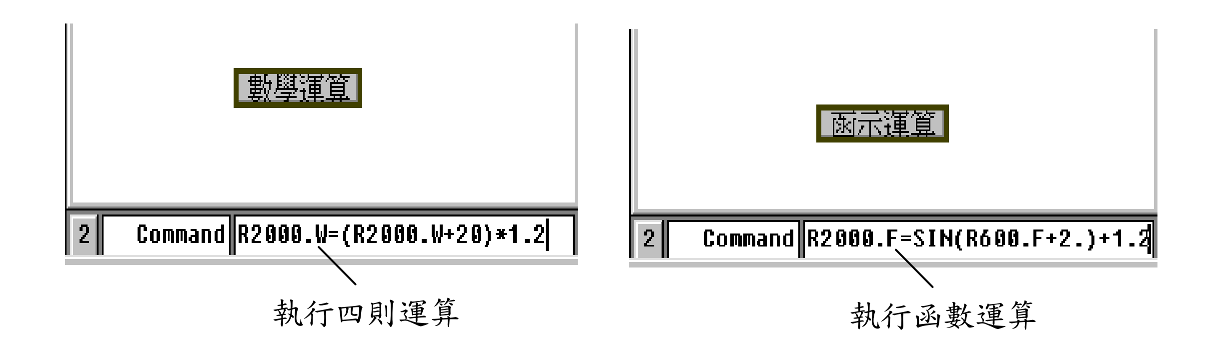
與「H-Scroll」相同功能，只是捲軸改為垂直式顯示罷了！所有設定模式均相同，不再贅述。

Button —**多用途之按鍵**（輸入/輸出型）

此項功能就是設計一個按鍵，而且可以由使用者自行定義按鍵的長度及按鍵上的文字（在"Title"欄內設定，中英文皆可 ），只要按下此按鍵就可以執行在其「元件組成變數」－"Command"欄內所定義的"運算式"，此運算式包含fPLC的所有「直接指令」、數學運算子、各種函式、檔案呼叫(CAI、PLC）等，而且可以連用兩個運算式以上，中間以雙引號（;;）區隔即可。功能非常強大，若能靈活應用，對於程式之設計有很大的幫助。以下列出常用的幾個例子：



執行「直接指令」　　呼叫「PLC」程式



步驟1：

1.按【Draw】功能表

2.選【Button】功能 (指標變成十字形)

3.移動指標到工作區所要繪製的地方

4.按下滑鼠左鍵以「拖曳」的方式繪製一長方形外框，然後釋放滑鼠左鍵，就會產生一個標有「Test」的按鍵（內定的文字）

步驟2：

1.按【Draw】功能表

2.選【 】功能

3.指到剛才所繪的「Test」按鍵處，並按一下滑鼠左鍵→視窗底部兩欄位分別出現"Title"、"Test"字樣。

4.將滑鼠指到視窗最底下「Test」處，並按一下左鍵以〝←〞(back space鍵)去掉〝Test〞文字，然後輸入適當的按鍵名稱，然後按【Enter】鍵。

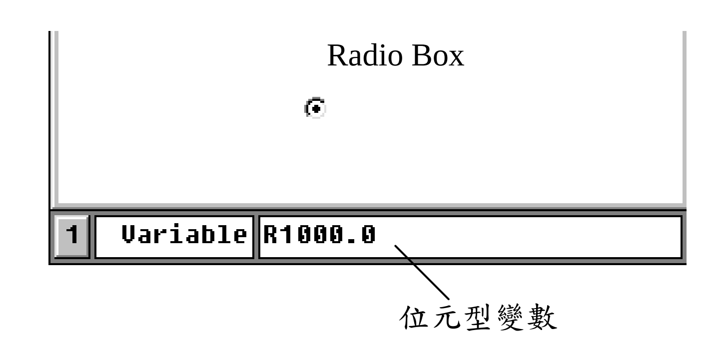
5.按一下〝↓〞，在視窗底部出現"Command"字樣，將滑鼠指到旁邊「物件變數設定欄」內並按左鍵，出現輸入模式

的閃爍直線。

6.輸入欲執行的運算式或命令，並按【Enter】鍵，即完成設定。

Radio Box — **圓形按鈕/燈號**（輸入/輸出型）

將變數以外型似”收音機按鈕”的圓形鈕來表示，此類元件**只可適用於「位元（bit）形態」的變數**，如R1000.0、R200.6等，也就是變數值可能為「1」或「0」（以”開關”的觀念來看就是”ON”或”OFF”），當所對應的變數值為1時，圓形鈕中間有一黑點；為0時，圓形鈕中間為空白。此元件可用於ON/OFF形式的開關或顯示燈號，如下圖：



步驟1：

1.按【Draw】功能表

2.選【Radio Box】功能（指標變成十字形）

3.移動指標到工作區所要設定的地方

4.按下滑鼠左鍵後即可出現一個圓形按鈕

步驟2：

1.按【Draw】功能表

2.選【 】功能

3.指到剛才所設的圓形按鈕處，並按一下滑鼠左鍵→視窗底

部兩欄位分別出現"Variable"、"R1000.0"字樣。

4.將滑鼠指到視窗最底下「R1000.0」處，並按一下左鍵以〝←〞(back space鍵)去掉〝R1000.0〞文字，然後輸入適當的**位元變數名稱**，然後按【Enter】鍵，即完成設定。

Check Box — **方形檢查按鈕/燈號** (輸入/輸出型)

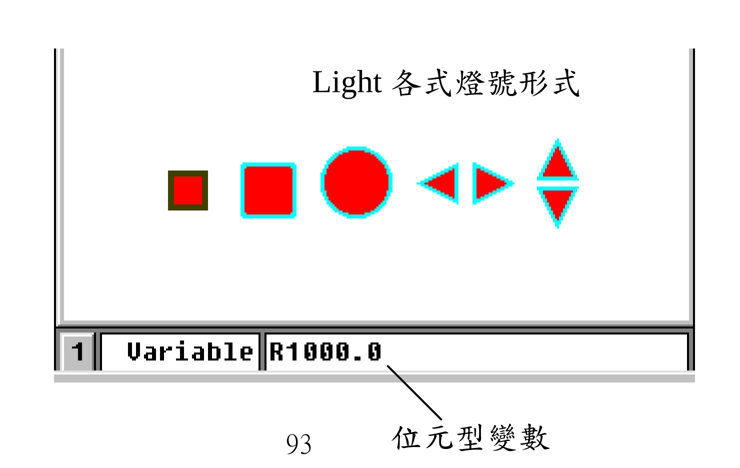
與「Radio Box」相同功能，只是外型改為方形，中間的黑點改成「打勾記號」罷了！所有設定方式均相同，不再贅述。

以下所介紹的，屬於第三項－**只可單向輸出型的**「動態變數資料物件」

Light — **多樣式燈號** (輸出型－元件外型可改變)

本元件之功能與「Radio Box」、「Check Box」類似，也是將**「位元(bit)形態」的**變數（如：R1000.0）以燈號的外型來表示，只是此類元件的外型大小、顏色可以由User自行改變，比前面所述的元件較為生動。透過滑鼠對元件的「拖曳控制」可以調整燈號外型的大小，透過功能項「Object」下的子功能「Fill」、「Line」、「Data Shape」等，可以改變元件的背景顏色、元件作動時顏色、元件的形狀等。

本元件內定為方形，可由「Data Shape」改成其他形狀。當所對應的變數值(bit型)為1時(表示此時為"作動")，元件的顏色變成由「Line」功能所設定的"作動顏色"；變數值為0時，元件的顏色變成由「Fill」功能所設定的"背景顏色"。各式燈號形狀，如下圖所示：



步驟1：

1.按【Draw】功能表

2.選【Light】功能（指標變成十字形）

3.移動指標到工作區所要設定的地方

4.按下滑鼠左鍵後，以「拖曳」的方式繪製一長方形外框，然後釋放滑鼠左鍵，就會產生一個兼具「背景顏色」與「作動顏色」的方型燈號

步驟2：

1.按【Draw】功能表

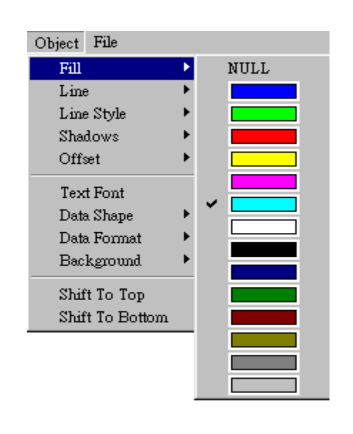
2.選【 】功能

3.指到剛才所設的燈號處，並按一下滑鼠左鍵→視窗底部兩欄位分別出現”Variable”、”R1000.0”字樣。

4.將滑鼠指到視窗最底下「R1000.0」處，並按一下左鍵以〝←〞(back space鍵)去掉〝R1000.0〞文字，然後輸入適當的**位元變數名稱**，然後按【Enter】鍵，即完成設定。

步驟3：

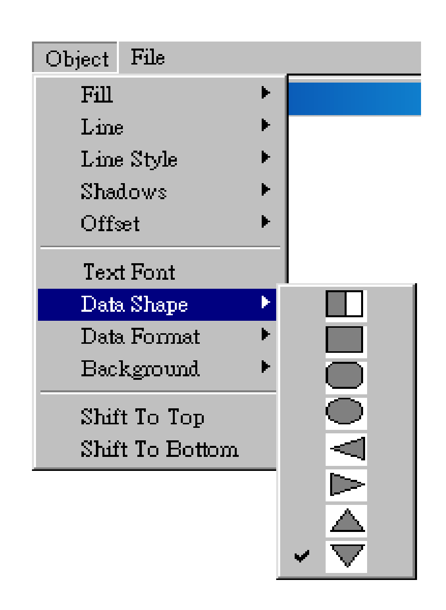
如果想要改變燈號的顏色，請將滑鼠指到「Object」，然後選擇「Fill」，會出現一個「顏色選擇表」，將滑鼠指到想要的顏色上並按一下（註：”NULL”表示不塗顏色），即可改變



元件的「背景顏色」；如果要改變元件的「作動顏色」，可以選擇「Line」用同樣的方式來設定。如下圖所示：

步驟4：

如果想要改變燈號的形式，請將滑鼠指到「Object」，然後選擇「Data Shape」，會出現一個「元件形式選擇表」，將滑鼠指到想要的形式上並按一下，即可改變元件的形式。如下圖所示：

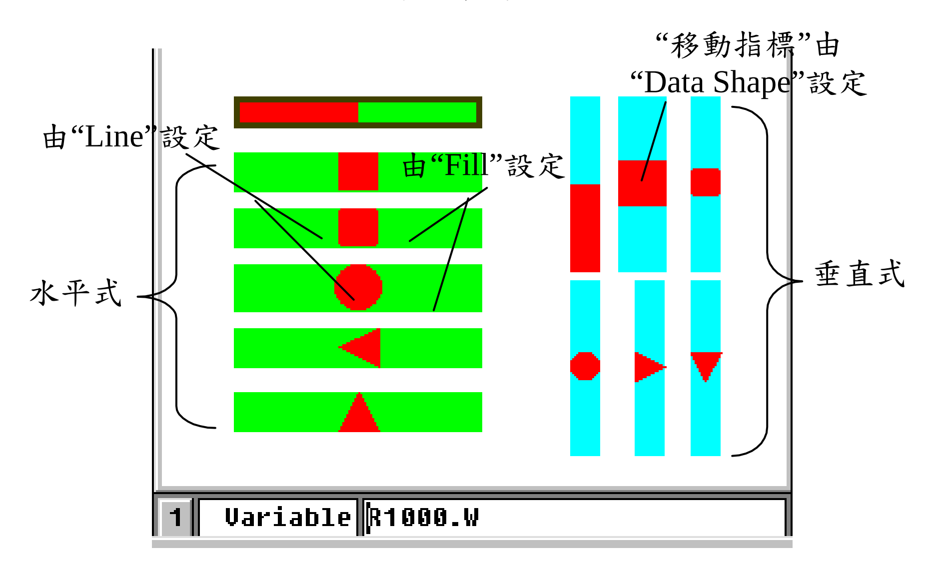


Level — **水銀柱式顯示器** (輸出型－元件外型可改變)

本元件之功能與「H-Scroll」、「V-Scroll」類似，但是**只可當成顯示的元件（無法由滑鼠移動指標來改變數值）**。將變數值以〝**水銀柱**〞移動的方式來表示。此類元件適用於整數及浮點數（具小數點），具有兩組「元件組成變數」－『Minimum』、『Maximum』，分別用來設定左極限（指標移至最左邊時）及右極限（指標移至最右邊時）之數值，內定值分別為0、100。此元件除了可擺成水平外，也可擺成垂直

形式，並且可改變其元件寬度及「移動指標」的形式（由「Data Shape」功能項來改變），其外型如下圖所示：

各式水銀柱形顯示器



步驟1：

1.按【Draw】功能表

2.選【Level】功能（指標變成十字形）

3.移動指標到工作區所要設定的地方

4.按下滑鼠左鍵後，以「拖曳」的方式繪製一長方形外框，然後釋放滑鼠左鍵，就會產生一個兼具「背景顏色」與「指標顏色」的「水銀柱式顯示器」

步驟2：

1.按【Draw】功能表

2.選【 】功能

3.指到剛才所設的顯示器處，並按一下滑鼠左鍵→視窗底部兩欄位分別出現"Variable"、"R1000"字樣。

4.將滑鼠指到視窗最底下「R1000」處，並按一下左鍵以〝←〞

(back space鍵)去掉〝R1000〞文字，然後輸入適當的變**數名稱**，然後按【Enter】。

5.按一下〝↓〞，在視窗底部出現"Minimum"字樣，將滑鼠指到「物件變數設定欄」內並按左鍵，出現輸入模式的閃爍直線。

6.將內定值0刪去，再輸入左極限（下極限）之數值，並按【Enter】鍵。

7.再按一下〝↓〞，出現"Maximum"字樣，將內定值100刪去，再輸入右極限（上極限）之數值，並按【Enter】鍵，即完成設定。

步驟3：

如果想要改變顯示器的顏色，可依前項所述的方式，以「Object」功能下的「Fill」功能來改變元件的「背景顏色」；以「Line」功能來改變元件的「指標顏色」，不再贅述。

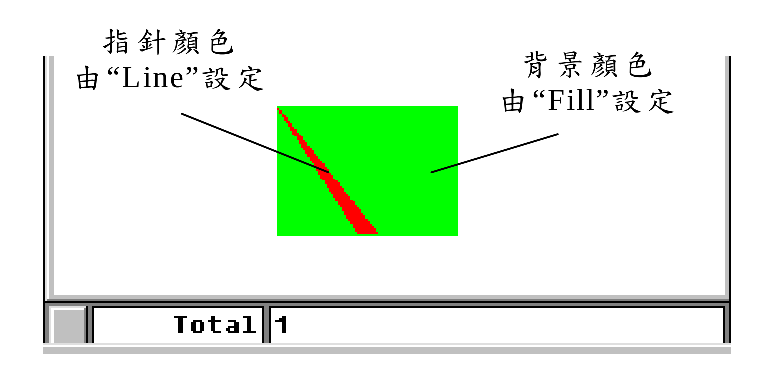
步驟4：

如果想要改變顯示器的「指標形式」（會移動的部份），可選擇「Object」功能下的「Data Shape」來設定，如前項所述，不再贅述。

Meter — **指針式顯示器** (輸出型－元件外型可改變)

本元件將變數值以〝指針式〞偏移的方式來表示（如類比式電表）。此類元件適用於整數及浮點數（具小數點），具有兩組「元件組成變數」－『Minimum』、『Maximum』，分別用來設定左極限（指針指到最左邊時）及右極限（指針指到最右邊時）之數值，內定值分別為0、100。可改變指針的顏色、背景顏色等，其外型如下圖所示：

指針形顯示器



步驟1：

1.按【Draw】功能表

2.選【Meter】功能 (指標變成十字形)

3.移動指標到工作區所要設定的地方

4.按下滑鼠左鍵後，以「拖曳」的方式繪製一長方形外框，然後釋放滑鼠左鍵，就會產生一個兼具「背景顏色」與「指針顏色」的「指針式顯示器」

步驟2：

1.按【Draw】功能表

2.選【 】功能

3.指到剛才所設的顯示器處，並按一下滑鼠左鍵→視窗底部兩欄位分別出現”Variable”、”R1000”字樣。

4.將滑鼠指到視窗最底下「R1000」處，並按一下左鍵以〝←〞(back space鍵)去掉〝R1000〞文字，然後輸入適當的變**數名稱**，然後按【Enter】。

5.按一下〝↓〞，在視窗底部出現”Minimum”字樣，將滑鼠指到「物件變數設定欄」內並按左鍵，出現輸入模式的閃爍直線。

6.將內定值0刪去，再輸入左極限之數值，並按【Enter】鍵。
7.再按一下〝↓〞，出現"Maximum"字樣，將內定值100刪去，再輸入右極限之數值，並按【Enter】鍵，即完成設定。

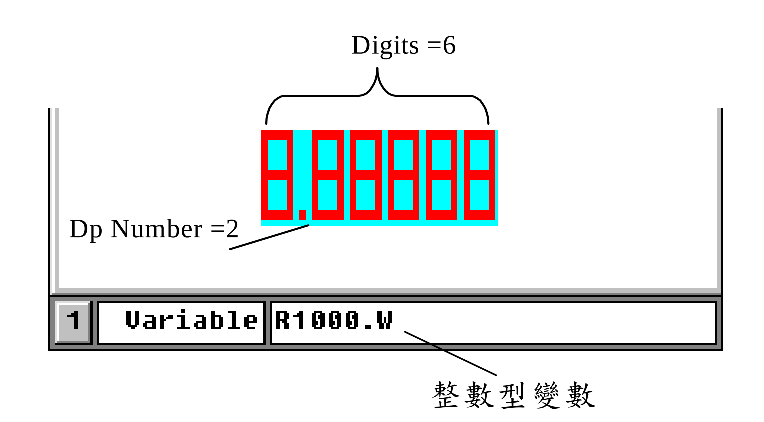
步驟3：

如果想要改變顯示器的顏色，可依前項所述的方式，以「Object」功能下的「Fill」功能來改變元件的「背景顏色」；以「Line」功能來改變元件的「指針顏色」，不再贅述。

Digit — **數字型顯示器** (輸出型－元件外型可改變)

本元件將變數值以單純的〝數字〞方式來表示，每位數字都有〝七段式顯示器〞般的顯示效果，雖然是顯示數字，但是其數字大小、顏色、背景顏色，顯示的位數、小數點位置等均可調整，比「Edit Box」元件的顯示效果靈活多了。不過，**此元件只有顯示功能，無法當輸入(設定)的元件，而且只能適用於整數型變數**。有一點須特別說明，「元件組成變數」中的『Digits』項是用來設定顯示的位數，而『Dp number』是用來設定顯示時小數點是否出現，或出現在哪個位置，若為0則不顯示小數點，若為1則出現在第1位數字的左邊(”第1位數字”指最左邊的數字)，為2則出現在第2位數字的左邊，依此類推。***千萬注意，此功能只是顯示的效果，絕對不代表變數本身的小數點位置***，因為此元件只能適用整數型變數，根本沒有小數點的問題。若容易搞混，建議『Dp number』值均設為0即可。

數字式顯示器



步驟1：

1.按【Draw】功能表

2.選【Digit】功能（指標變成十字形）

3.移動指標到工作區所要設定的地方

4.按下滑鼠左鍵後，以「拖曳」的方式繪製一長方形外框，然後釋放滑鼠左鍵，就會產生一個兼具「背景顏色」與「數字顏色」的「數字型顯示器」（內定為兩位數字）

步驟2：

1.按【Draw】功能表

2.選【 】功能

3.指到剛才所設的顯示器處，並按一下滑鼠左鍵→視窗底部兩欄位分別出現"Variable"、"R1000"字樣。

4.將滑鼠指到視窗最底下「R1000」處，並按一下左鍵以〝←〞(back space鍵)去掉〝R1000〞文字，然後輸入適當的變數名稱（**整數型**），然後按【Enter】。

5.按一下〝↓〞，在視窗底部出現"Digits"字樣，將滑鼠指到「物件變數設定欄」內並按左鍵，出現輸入模式的閃爍直線。

6.將內定值2刪去，再輸入欲顯示的數字位數之數值，並按【Enter】鍵。

7.再按一下〝↓〞，出現”Dp number”字樣，若要設定小數點之顯示，將內定值0刪去，再輸入小數點出現的位置，並按【Enter】鍵，即完成設定。

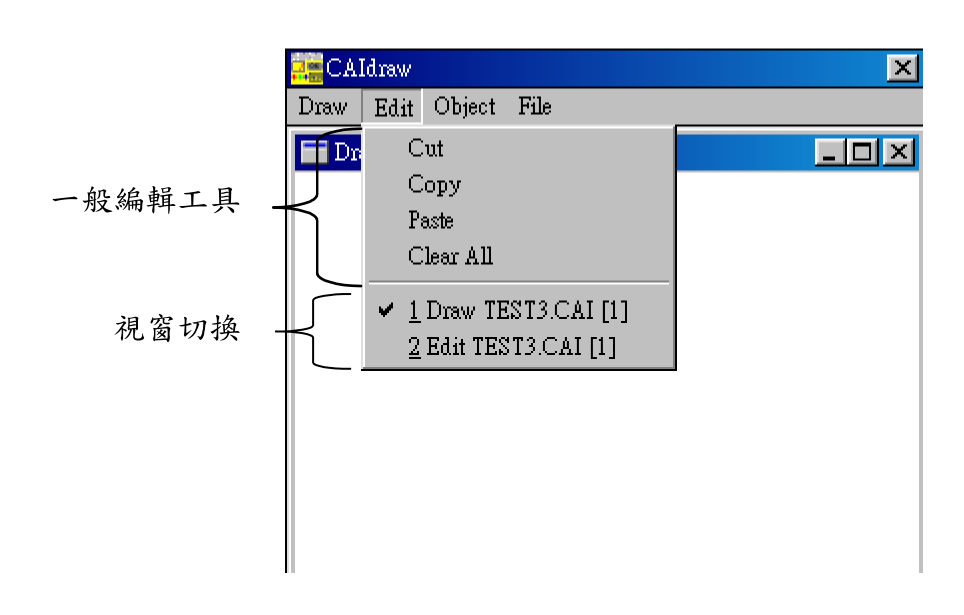
步驟3：

如果想要改變顯示器的顏色，可依前項所述的方式，以「Object」功能下的「Fill」功能來改變元件的「背景顏色」；以「Line」功能來改變元件的「數字顏色」，不再贅述。

3.4 『Edit』—編輯工具

CAIdraw視窗的第二群功能表—『Edit』，相信大家一定很眼熟，就如很多Windows軟體所提供的編輯工具一樣，主要的功能是針對前一節所提的控制元件執行一些常見的編輯工作，使你在撰寫圖控程式時可以更方便、更靈活。縮短開發程式的時間。

當你進入CAIdraw視窗後，以滑鼠左鍵點選一下「Edit」功能表單，就會出現以下畫面：



「Edit」功能表以橫隔線分成兩區，如上圖所標示，其中：

第1項屬於「一般編輯工具」，與一般軟體之編輯工具類似，可針對單一元件或元件群執行「Cut」、「Copy」、「Paste」、「Clear All」等編輯功能。

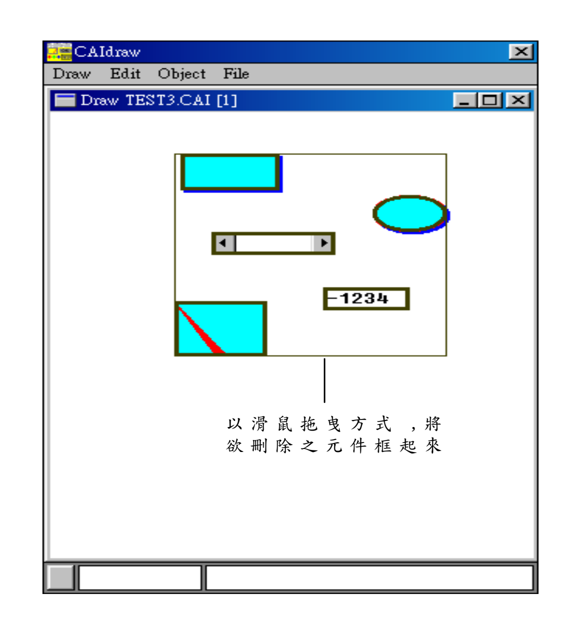
第2項則為「視窗切換」，可切換CAIdraw下的『Draw視窗』或『Edit視窗』。通常是設在『Draw視窗』的模式下，『Edit視窗』是針對一些特殊用途或須以「指令形式」來撰寫的「CAI」指令，一般使用者不會用到此視窗，所以在本手冊也不打算介紹。

Cut — **剪下 （刪除）**

本功能可針對所選擇的元件或元件群執行〝剪下〞的功能，也就是刪除以滑鼠所選到的元件，使其從工作區上消失。但是系統會自動將這些資料存在「剪貼簿」暫存區中，可以用『Paste』功能將其〝貼回〞工作區上，而且可重複、多次地〝貼回〞（即複製）。

步驟1：

1. 按【Draw】功能表
2. 選【 】功能 （指標變成箭頭）
3. 以箭頭指到想刪除的元件上然後按下左鍵或以「拖曳」的方式將元件包含於框框內，如下圖：



步驟2：

1.以滑鼠選擇【Edit】功能表
2.按一下【Cut】，則所選的元件馬上消失，完成刪除工作

Copy — 複製

本功能可針對所選擇的元件或元件群執行〝複製〞的功能，也就是將所選的元件資料存於「剪貼簿」暫存區中，稍後便可以用『Paste』功能將其完整地貼到工作區上，而且可執行多次。

步驟1：

1.按【Draw】功能表
2.選【 】功能 （指標變成箭頭）
3.以箭頭指到想刪除的元件上然後按下左鍵或以「拖曳」的方式將元件包含於框框內

步驟2：

1.以滑鼠選擇【Edit】功能表
2.按一下【Copy】，則所選的元件資料馬上存於「剪貼簿」暫存區中，可供【Paste】使用

Paste — 貼回

本功能可針對目前存在「剪貼簿」暫存區中的元件，執行〝貼回〞的動作。也就是將以【Cut】、【Copy】方式所存的元件再複製一份貼回工作區，而且每按一次【Paste】即在工作區產生一份相同元件。須注意一點，貼回的元件均落在相同位置，所以應該以滑鼠將其移開原來位置，以免元件重疊。

步驟：

1. 以滑鼠選擇【Edit】功能表
2. 按一下【Paste】，「剪貼簿」暫存區中的元件，即出現在工作區中
3. 以滑鼠移開所貼回的元件，以免元件重疊
4. 分別為元件編上新的變數名稱及「元件組成變數」的設定

Clear All — 清除所有元件

本功能可一次刪除工作區中所有的元件，使其恢復成剛進入CAIdraw視窗時的情形，不需要以滑鼠來選元件。**注意！一但以『Clear All』刪除所有元件，就無法以『Paste』功能來恢復**。

步驟：

1. 以滑鼠選擇【Edit】功能表
2. 按一下【Clear All】，出現一個詢問框：

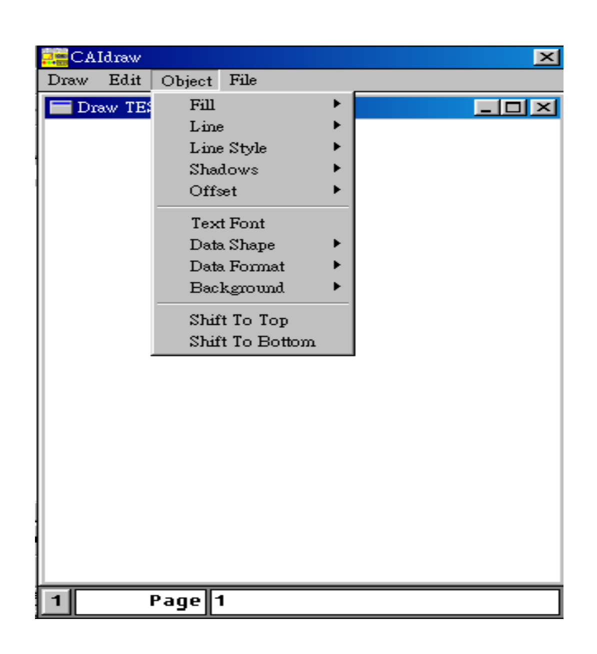


3. 按下『確定』鍵，全部元件均消失，無法復原

3.5 『Object』—物件設定工具

CAIdraw視窗的第三群功能表—『Object』，主要用途在設定由『Draw』所產生的元件的各項屬性特性，使得各元件不僅具備系統內定的樣式、型態、功能，而且可藉由『Object』下的各項子功能操作，得到更多的變化，使得圖控程式的設計更為生動，更具親和力。

當你進入CAIdraw視窗後，以滑鼠左鍵點選一下「Object」功能表單，就會出現以下畫面：



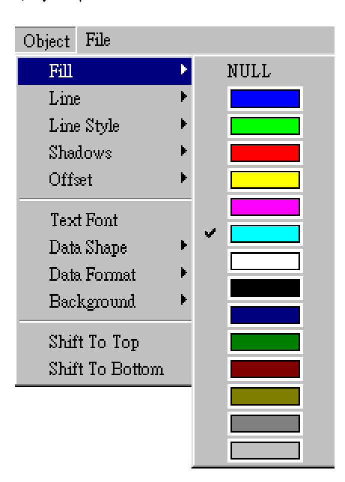
如上圖，『Object』共分成三組，以下一一介紹其用法：

Fill — **背景顏色之設定**

可改變「靜態圖形物件」的圖形顏色（塗滿部份）或「動態變數資料物件」（只限"輸出型"）的"背景顏色"。

步驟：

1.按【Draw】功能表→選【　】功能
2.以箭頭選取元件或以「拖曳」的方式將多個元件包含於框框內
3.按【Object】功能表→選【Fill】功能，出現一個「顏色選擇表」，如下圖所示：



4.以滑鼠指到「顏色選擇表」上所要設定的顏色，並按下滑鼠左鍵，即可完成顏色的設定

Line — 框線/作動顏色之設定

可改變「靜態圖形物件」的圖形外框顏色（含「Text」元件的文字顏色、線條元件的線條顏色等）或「動態變數資料物件」（只限"輸出型"）的"作動顏色"（隨變數值變動的部份）。

步驟：

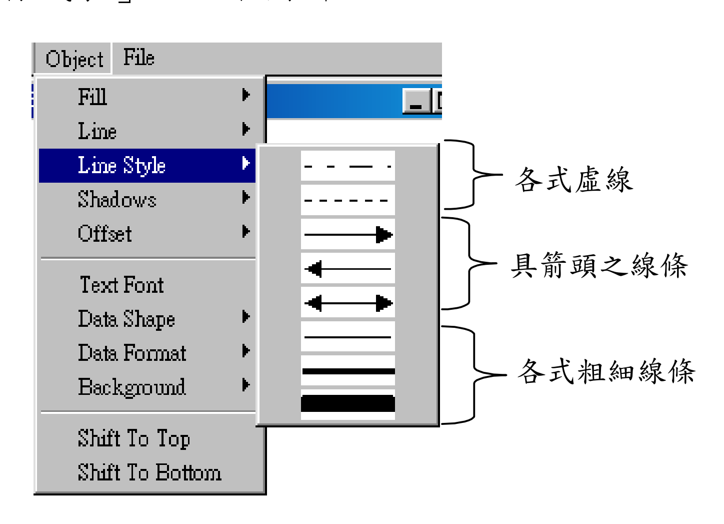
1.按【Draw】功能表→選【　】功能
2.以箭頭選取元件或以「拖曳」的方式將多個元件包含於框框內
3.按【Object】功能表→選【Line】功能，出現一個「顏色選擇表」，如前項（「Fill」）所示。
4.以滑鼠指到「顏色選擇表」上所要設定的顏色，並按下滑鼠左鍵，即可完成顏色的設定

Line Style — 線條樣式之設定

用來改變「線條元件」所繪出線條的樣式或「圖形元件」的外框線樣式。

步驟：

1. 按【Draw】功能表→選【 】功能
2. 以箭頭選取元件或以「拖曳」的方式將多個元件包含於框框內
3. 按【Object】功能表→選【Line Style】功能，出現一個「線條樣式表」，如下圖所示：



4. 以滑鼠指到「線條樣式表」上所要設定的樣式，並按下滑鼠左鍵，即可完成線條樣式的設定

Shadows — 圖形元件之陰影顏色設定

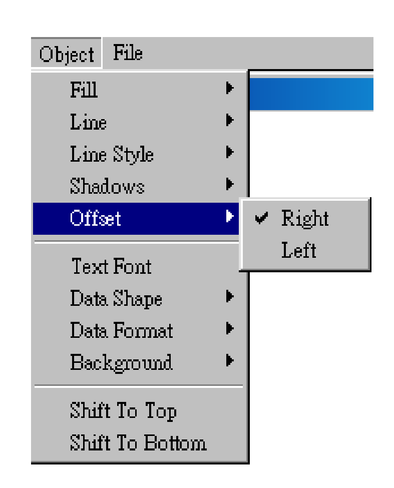
用來改變「圖形元件」所繪出圖形的陰影顏色。其設定步驟與前述「Fill」、「Line」等類似，不再贅述。不過，如果要將圖形元件之陰影去除，就要在「顏色選擇表」上選擇「NULL」這一項。

Offset — 圖形元件之陰影顏色設定

用來改變「圖形元件」陰影效果所出現的地方，可置於圖形之左上角（Left）或右下角（Right）。

步驟：

1.按【Draw】功能表→選【 】功能
2.以箭頭選取元件或以「拖曳」的方式將多個元件包含於框框內
3.按【Object】功能表→選【Offset】功能，出現一個「Offset位置表」，如下圖所示：



4.以滑鼠指到「Offset位置表」上所要設定的位置，並按下滑鼠左鍵，即可完成陰影Offset的設定

Text Font — 文字元件之字體設定

用來設定「Text」元件（文字元件）所寫文字的字體樣式，本項功能與一般Windows軟體之字型設定方式類似。

步驟：

1.按【Draw】功能表→選【 】功能
2.以箭頭選取文字元件或以「拖曳」的方式將多個文字元件包含於框框內

3.按【Object】功能表→選【Text Font】功能，出現一個「字型對話框」，如下圖所示：



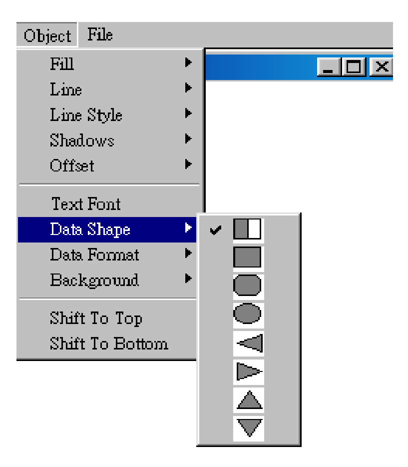
4.填好以上「字型對話框」內有關之字型資料，並按下「確定」鍵後即可完成

Data Shape — 「Light」形狀/「Level」移動指標形狀之設定

用來設定動態元件「Light」燈號的外型或「Level」顯示器的移動指標形式。

步驟：

1.按【Draw】功能表→選【 】功能
2.以箭頭選取「Light」/「Level」元件或以「拖曳」的方式將多個元件包含於框框內
3.按【Object】功能表→選【Data Shape】功能，出現一個「樣式選擇表」，如下圖所示：



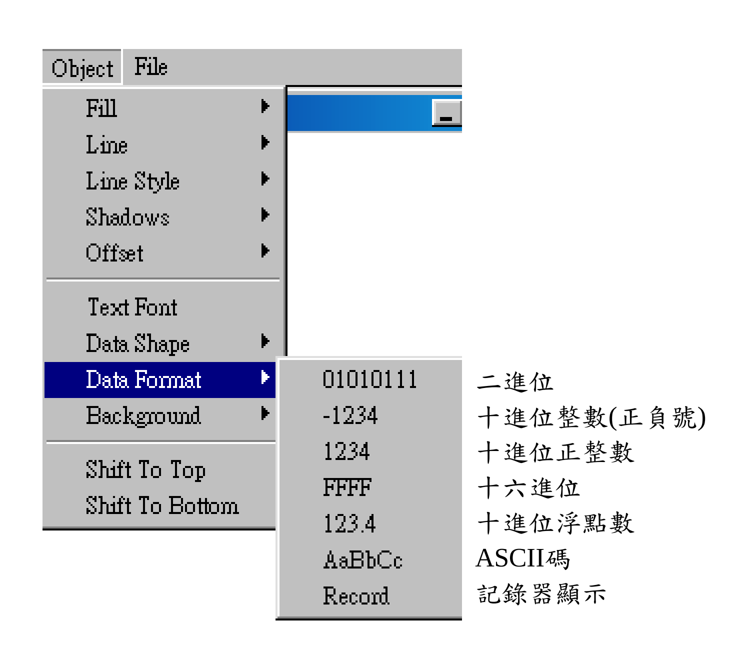
4. 以滑鼠指到「樣式選擇表」上所要設定的樣式，並按下滑鼠左鍵，即可完成樣式的設定

Data Format — 「Edit Box」數字顯示模式之設定

用來設定動態元件「Edit Box」的數字顯示模式(數字系統的基底)，如「二進位」、「十進位整數」、「十進位浮點數」、「十六進位」、「ASCII碼」、「記錄器顯示」等。

步驟：

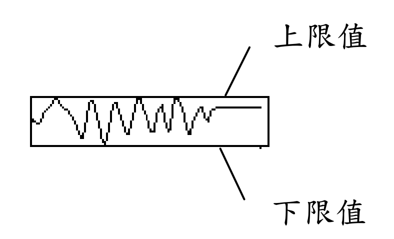
1. 按【Draw】功能表→選【 】功能
2. 以箭頭選取「Edit Box」元件或以「拖曳」的方式將多個元件包含於框框內
3. 按【Object】功能表→選【Data Format】功能，出現一個「數字系統表」，如下圖所示：



4. 以滑鼠指到「數字系統表」上所要設定的項目，並按下滑鼠左鍵，即可完成數字系統的設定

特別注意！如果選擇最後一項「Record」－記錄器顯示方式，則須加上以下步驟：

5.當「Edit Box」元件出現「Record」字樣後，將滑鼠隨便指到別處並按一下左鍵，然後再指到剛才處理的「Edit Box」元件
6.按一下〝↓〞，在視窗底部出現”Minimum”字樣，將滑鼠指到「物件變數設定欄」內並按左鍵，出現輸入模式的閃爍直線。
7.輸入記錄器顯示框的「下限值」，並按【Enter】鍵。
8.再按一下〝↓〞，出現”Maximum”字樣，輸入記錄器顯示框的「上限值」，並按【Enter】鍵，才算完成。

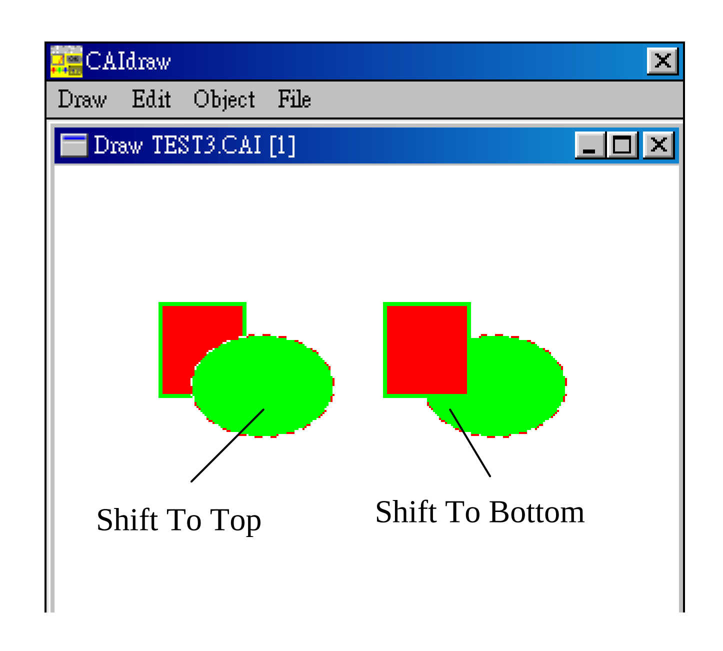


Background — 工作區背景顏色之設定

用來設定圖控程式工作區底稿之背景顏色。一但以此功能更改顏色後，則整個工作區均為所設定的顏色，但不會影響其他元件，設定方式與「Fill」、「Line」相同，不再贅述。

Shift To Top 與Shift To Bottom— 重疊元件之上下順序設定

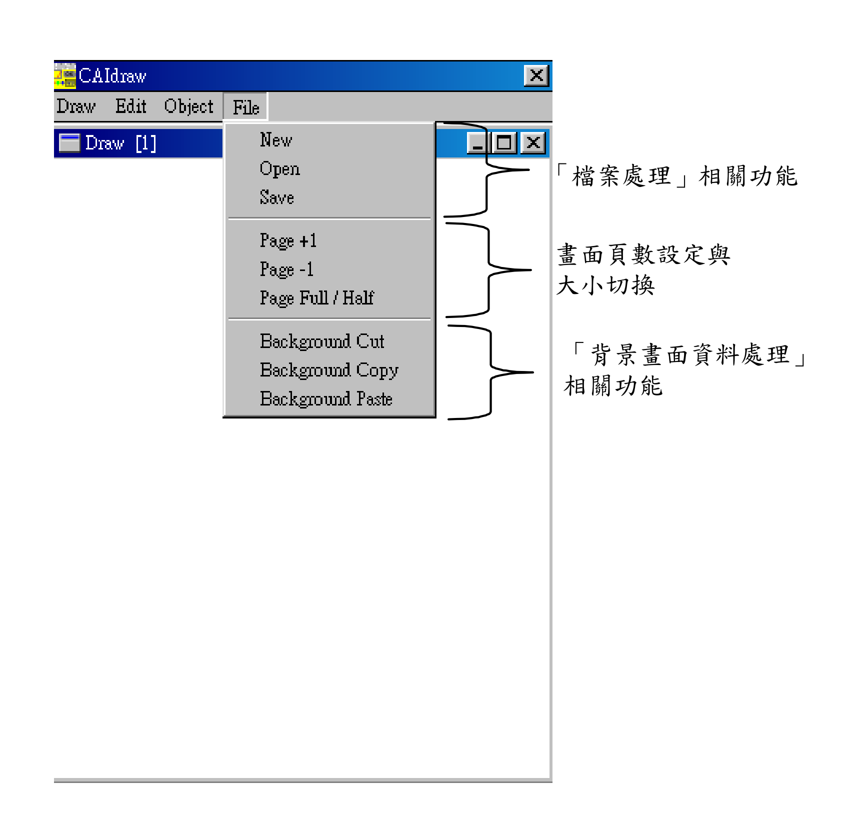
前者將重疊的元件擺至最上面，重疊處可以看得到；後者將將重疊的元件擺至最下面，重疊處無法看到。對於處理多個元件擠在一起的情形，可以用滑鼠指到要處理的元件，然後按下「Shift To Top」或「Shift To Bottom」，系統即會自動重排。如下圖說明：



3.6『File』— 檔案處理

CAIdraw視窗的第四群功能表—『File』，以分隔線分成三個部份，第一部份用來處理與檔案有關的事情，第二部份與CAI畫面頁數設定及畫面大小切換有關，第三部份則與背景畫面資料之處理有關，詳述如下：

當你進入CAIdraw視窗後，以滑鼠左鍵點選一下「File」功能表單，就會出現以下畫面：



New — 開新檔案

本功能與一般文書編輯軟體上的「開啟新檔」功能一樣，也就是重新開啟一個空白的CAI檔案，如果你現在正在編輯一個檔案，但是想放棄所有編輯的元件時，就可以用「New」的功能，將所有元件清除，以一個新的檔名再重新編輯。當你選擇「New」時，會出現一個「new CAI file」的詢問框，如下圖所示：



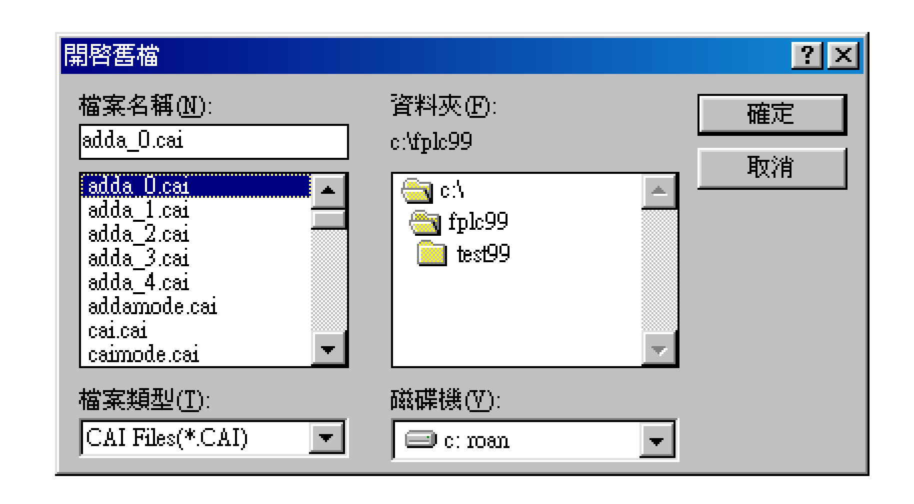
只要按下『確定』鍵即可將目前所修改的部份全部放棄（不執行存檔

動作）然後呈現一個空白的CAI畫面，供你重新編寫。當然！你也可以按下「取消」鍵維持原來編寫的內容，不作任何更動。

記得一件事，當你以「New」設定一個新檔並編寫完畢後，要以「Save」功能將其存檔，並給予一個新的檔案名字，否則所有的編寫將無效。

Open — 開啟舊檔案

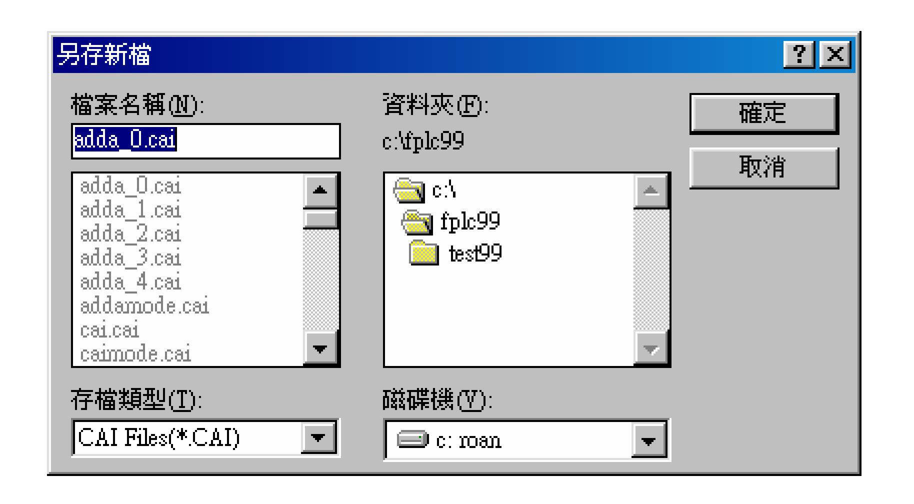
本功能可開啟原先已經存在的舊檔案，供你觀察或修改，甚至存成別的檔案名字。當你按下「Open」功能項後，也會出現一個「開啟舊檔」對話框，如下：



你可以用滑鼠點選其中的檔案，或者直接在「檔案名稱」方框內輸入檔名，然後按「確定」鍵，即可開啟舊有CAI檔，並呈現在CAI工作區上。有一點須特別強調，因為一個CAI檔可能有一頁以上的畫面，如果沒有特別指定「頁碼」（設定方法是用「Page+1」、「Page-1」功能），系統內定是叫出第一頁的資料，如果想叫出其它頁的內容，則須先在執行「Open」功能之前，以「Page+1」或「Page-1」設定頁碼（CAI之「視窗訊息顯示」處會顯示目前頁碼），如此便可叫出所設定頁碼的資料並顯示在畫面上。

Save — 儲存檔案

本功能可將正在編寫的CAI檔儲存起來，或者存成新的檔案，按下「Save」功能時，會出現一個「另存新檔」對話框，如下：



在「檔案名稱」的方框內會顯示目前的檔名，只要按下「確定」鍵，然後在出現的一個「警告對話框」（如下圖）內按下「是」鍵，即可將目前所編寫、修改的內容以原來檔名存檔。

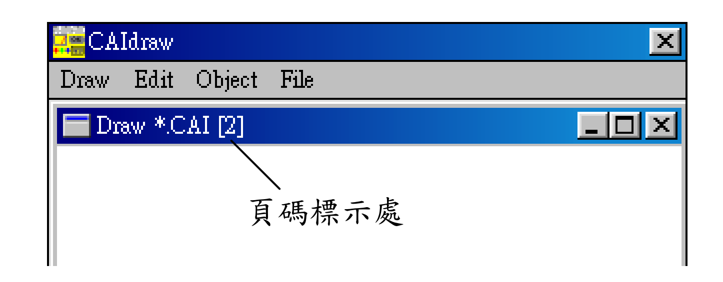


如果要存成新的檔案，只要在「檔案名稱」方框內輸入檔名，然後再按「確定」鍵即可，如此就不會更改到原來的檔案資料。

Page+1 與 Page-1 — 頁碼設定

本功能可設定CAI檔的頁碼，當你按一次「Page+1」時，在CAI之「視窗訊息顯示」處的頁碼數字就會加一，按一次「Page-1」時，頁碼數

字就會減一，如下圖：



頁碼的設定，除了前面提到可以在「Open」前設定所要開啟的頁次外，**也可以用來將現有頁次的畫面資料存到其他頁次**。例如在某頁次的編輯中（如：Page 1），以「Page+1」或「Page-1」設定成其它的頁次（如：Page 2，「視窗訊息顯示」處的頁碼數字變成2），然後以「Save」功能來將修改後的資料存到CAI檔的另一頁（Page 2），而不會更改原先（Page 1）的內容。

Page Full/Half — **CAI全頁/半頁畫面切換**

本功能可以用來切換CAI畫面的顯示模式，系統內定為「半頁」模式，佔著螢幕右半邊；若按一下此功能項，則切換成「全頁」模式，佔掉整個螢幕，再按一次則恢復「半頁」模式。

4. 組合語言指令說明

DSP中的組合語言以簡短易讀為目標，所以指令數目非常少，共有29組。依功能可概分成下面幾類：

資料設定指令	RD	WR		
數值運算指令	ADD	ADC	SUB	SBC
數值運算指令	MUL	NEG	ABS	SGN
移位運算指令	SFR	SFL		
邏輯運算指令	AND	OR	XOR	COM
運算保護指令	MAX	MIN	PRT	
堆疊控制指令	PUSH	POP		
程式控制指令	JMP	JR	CALL	RET
程式控制指令	NOP	HALT		
迴圈控制指令	RPT	LOOP		

本章中的組合語言定義都依照字母順序編排，以方便查詢參考。

ABS指令

ABS指令是取絕對值的動作，其機械碼定義為

	15 13	12 9	8 4	0	
ABS r	011	func=10	0000	reg=0~31	k3類群
ABS	100	func=10	mode=4	00000	k4類群

ABS指令有兩種模式，就是

ABS; ;AX=abs(AX)的取絕對值動作。
ABS r; ; r=abs(r) 的取絕對值動作。

每個指令的細部分解動作說明如下：

ABS;	#1.PC++;	
clk=1	#2.AX=abs(AX)	運算動作，無重複控制

ABS reg;	#1.PC++; DP=MP+LC+reg+0x800	LC為迴圈參數，預設=0
clk=4	#2.AX=(DP)	讀入動作
	#3.AX=abs(AX)	運算動作
	#4.(DP)=AX	寫出動作，無重複控制

ADC指令

ADC指令是加法運算的動作。其機械碼定義為

	15　13	12　9	8　4	0	
ADC #k	000	func=7	k=+/-255		k0類群
ADC #k,bit	001	func=7	bit=0~15	k=0~31	k1類群
ADC r,bit	010	func=7	bit=0~15	reg=0~31	k2類群
ADC r,#k	011	func=7	k=0~15	reg=0~31	k3類群
ADC (r)	100	func=7	mode	reg=0~31	k4類群

其中mode=0~15，而根據mode的不同，包括了下面的多種模式：

```
mode=0 ADC.0  r;   ;AX=AX+ r +CF，r屬於公用變數區。
mode=1 ADC.0 (r);  ;AX=AX+(r)+CF，r屬於公用變數區。
mode=2 ADC   r;    ;AX=AX+ r +CF，r屬於模組變數區，而CF是進位旗標。
mode=3 ADC (r);    ;AX=AX+(r)+CF，r屬於模組變數區。
mode=4 ADC.H;      ;AX=AX+ HX +CF。
mode=5 ADC.M;      ;AX=AX+(HX)+CF。
mode=6 ADC.S  r;   ;先執行PUSH.A的儲存動作後，再執行ADC  r 指令。
mode=7 ADC.S (r);  ;先執行PUSH.A的儲存動作後，再執行ADC (r)指令。
mode=8 ADC.X  r;   ;執行執行ADC  r 的同時，將AX儲存至 r 中。
mode=9 ADC.X (r);  ;執行執行ADC (r)的同時，將AX儲存至(r)中。
mode=10 ADC.S;      ;直接從堆疊器取出資料來執行ADC運算。
```

而每個指令的細部分解動作說明如下：

```
ADC.0  reg;   #1.PC++; DP=reg                 取得公用變數區的位址
clk=1+n       #2.AX=AX+(DP)+CF                實際運算動作
              #3.DP++; if ((--RPT)>0) loop#2  此項配合重複指令控制
```

```
ADC.0 (reg);  #1.PC++; DP=reg                 取得公用變數區的位址
clk=2+n       #2.DP=(DP)                      間接讀寫動作
              #3.AX=AX+(DP)+CF                實際運算動作
              #4.DP++; if ((--RPT)>0) loop#3  此項配合重複指令控制
```

```
ADC    reg;   #1.PC++; DP=MP+LC+reg+0x800     LC爲迴圈參數，預設=0
clk=1+n       #2.AX=AX+(DP)+CF                實際運算動作
              #3.DP++; if ((--RPT)>0) loop#2  此項配合重複指令控制
```

```
ADC   (reg); #1.PC++; DP=MP+LC+reg+0x800    LC為迴圈參數，預設=0
clk=2+n      #2.DP=(DP)                     間接讀寫動作
             #3.AX=AX+(DP)+CF               實際運算動作
             #4.DP++; if ((--RPT)>0) loop#3 此項配合重複指令控制
```

```
ADC.H;       #1.PC++;
clk=1        #2.AX=AX+HX+CF                 實際運算動作，無重複控制
```

```
ADC.M;       #1.PC++;
clk=2+n      #2.DP=(HX++)                   取得HX暫存器內容作為指標
             #3.AX=AX+(DP)+CF               實際運算動作
             #4.DP++; if ((--RPT)>0) loop#3 此項配合重複指令控制
```

```
ADC.S reg;  #1.PC++; DP=MP+LC+reg+0x800    LC為迴圈參數，預設=0
clk=2+n     #2.(SP++)=AX                   PUSH.A動作
            #3.AX=AX+(DP)+CF               實際運算動作
            #4.DP++; if ((--RPT)>0) loop#3 此項配合重複指令控制
```

```
ADC.S (reg); #1.PC++; DP=MP+LC+reg+0x800    LC為迴圈參數，預設=0
clk=3+n      #2.DP=(DP)                     間接讀寫動作
             #3.(SP++)=AX                   PUSH.A動作
             #4.AX=AX+(DP)+CF               實際運算動作
             #5.DP++; if ((--RPT)>0) loop#4 此項配合重複指令控制
```

```
ADC.X reg;  #1.PC++; DP=MP+LC+reg+0x800    LC為迴圈參數，預設=0
clk=4+3n    #2.HX=AX                       將AX儲存至HX中動作
            #3.AX=AX+(DP)+CF               實際運算動作
            #4.(DP)=HX                     將HX儲存至記憶體中
            #5.DP++; if ((--RPT)>0) loop#2 此項配合重複指令控制
```

```
ADC.X (reg); #1.PC++; DP=MP+LC+reg+0x800    LC為迴圈參數，預設=0
clk=5+3n     #2.DP=(DP)                     間接讀寫動作
             #3.HX=AX                       將AX儲存至HX中動作
             #4.AX=AX+(DP)+CF               實際運算動作
             #5.(DP)=HX                     將HX儲存至記憶體中
             #6.DP++; if ((--RPT)>0) loop#3 此項配合重複指令控制
```

```
ADC.S;      #1.PC++
clk=1+n     #2.AX=AX+(--SP)+CF              實際運算動作
            #3.DP++; if ((--RPT)>0) loop#2  此項配合重複指令控制
```

```
ADC reg,bit; #1.PC++; DP=MP+LC+reg+0x800     LC爲迴圈參數，預設=0
clk=1+n      #2.AX=AX+((DP)<<bit)+CF         實際運算動作
             #3.DP++; if ((--RPT)>0) loop#2 此項配合重複指令控制
```

```
ADC #k,bit;  #1.PC++;
clk=1+n      #2.AX=AX+(k<<bit)+CF            實際運算動作，無重複控制
```

```
ADC #k;      #1.PC++
clk=1+n      #2.AX=AX+k+CF                   實際運算動作，無重複控制
```

```
ADC reg,#k;  #1.PC++; DP=MP+LC+reg+0x800     LC爲迴圈參數，預設=0
clk=4        #2.AX=(DP)                      讀取動作
             #3.AX=AX+k+CF                   實際運算動作
             #4.(DP)=AX                      儲存動作，無重複控制
```

加法運算有兩種，ADD指令不處理前次的進位，運算為(AX=AX+HX)的動作。而ADC指令要處理前次的進位，運算為(AX=AX+HX+CF)的動作。

由上述的細部分析可看出，真正的運算只是(AX=AX+HX+CF)的動作而已，所有的不同的模式就是選擇不同的HX設定方式。其他的數值和邏輯運算都有類似的結構，只是運算式的項目不同而已。和ADC架構相似的指令包括：

數值運算指令	ADD	ADC	SUB	SBC
數值運算指令	MUL			
邏輯運算指令	AND	OR	XOR	
運算保護指令	MAX	MIN		

另外，許多運算指令在執行時都會更動HX暫存器的內容，因此除非是立刻被用到，不然千萬不要用HX暫存器來儲存資料。

ADD指令

ADD指令是加法運算的動作，其機械碼定義為

	15 13	12 9	8	4 0	
ADD #k	000	func=6	k=+/-255		k0類群
ADD #k,bit	001	func=6	bit=0~15	k=0~31	k1類群
ADD r,bit	010	func=6	bit=0~15	reg=0~31	k2類群
ADD r,#k	011	func=6	k=0~15	reg=0~31	k3類群
ADD (r)	100	func=6	mode	reg=0~31	k4類群

ADD指令中，除了運算動作改成(AX=AX+HX)之外，其他所有定義和執行細節都和ADC指令相同。

AND指令

AND指令是邏輯運算的動作，其機械碼定義為

	15 13	12 9	8 4	0	
AND #k	000	func=0	k=+/-255		k0類群
AND #k,bit	001	func=0	bit=0~15	k=0~31	k1類群
AND r,bit	010	func=0	bit=0~15	reg=0~31	k2類群
AND r,#k	011	func=0	k=0~15	reg=0~31	k3類群
AND (r)	100	func=0	mode	reg=0~31	k4類群

AND指令中，除了運算動作改成(AX=AX.and.HX)之外，其他所有定義和執行細節都和ADC指令相同。

CALL指令

CALL指令是呼叫副程式的動作。其機械碼定義為

	15 　　12	11 　　　　　　　　0	
CALL #k	1110	k=0~4095	J7類群

CALL指令的細部分解動作說明如下：

```
CALL k; #1. PC++;              //PC指到下一位址
clk=3   #2. (SP++)=PC;         //儲存到堆疊器中
        #3. PC=(PC+k)&0x0fff;  //PC指到副程式的相對位址
```

為了搭配模組化程式的執行，CALL指令中的k值指的是相對位移。所以在編譯成機械碼後，整個模組即使是搬移位置後仍能正確的執行。

k值的計算由編譯程式自動產生，使用者不用費心。其範圍是0~4095，所以能呼叫整個程式區的範圍。

COM指令

COM指令是作位元反向的邏輯動作，其機械碼定義為

	15 13	12 9	8 4	4 0	
COM r	011	func=12	0000	reg=0~31	k3類群
COM	100	func=12	mode=4	reg=0~31	k4類群

和ABS指令一樣，COM指令有兩種模式，就是

COM;　　;AX=~AX 的位元反向動作。

COM r;　; r=~r　的位元反向動作。

每個指令的細部分解動作說明如下：

COM;	#1.PC++;	
clk=1	#2.AX=~AX	運算動作，無重複控制

COM reg;	#1.PC++; DP=MP+LC+reg+0x800	LC爲迴圈參數，預設=0
clk=4	#2.AX=(DP)	讀入動作
	#3.AX=~AX	運算動作
	#4.(DP)=AX	寫出動作，無重複控制

HALT指令

HALT指令結束DSP的軟體動作，將控制權還給硬體。其機械碼定義為

	15 13	12 9	8 4	0	
HALT	101	func=0	0000	00000	N5類群

HALT指令由系統程式自動產生，在一般模組程式中不准使用。一旦系統程式用HALT指令做結束後，除非是硬體再度啟動@initial、@stop、@timer或是@doloop程序，否則軟體將一直維持在凍結的狀態。

JMP指令

JMP指令是絕對位址的跳躍的動作，其機械碼定義包括：

	15　　12	11　　　　　　0	
JMP #k	1101	k=0~4095	J7類群

JMP指令由系統程式自動產生，在一般模組程式中不准使用。通常系統程式是在軟體啟動後，用JMP指令跳到@initial、@stop、@timer或是@doloop程序的啟點，開始執行各個模組程式。

JMP指令的細部分解動作說明如下：

JMP #k;
clk=2
	#1.PC=k;	跳到指定位址繼續執行

JR指令

JR指令是依條件來改變程式的執行步驟。其機械碼定義為

	15 13	12 7	0	
JR #k	1100	cond	k=+/-127	J6 類群

其中cond=0~15。根據cond的不同，而有下面的多種模式：

```
cond=0  JR.C  #k; ;若CF=1則PC=PC+1+k，否則PC=PC+1。
cond=1  JR.NC #k; ;若CF=0則PC=PC+1+k，否則PC=PC+1。
cond=2  JR.Z  #k; ;若ZF=1則PC=PC+1+k，否則PC=PC+1。
cond=3  JR.NZ #k; ;若ZF=0則PC=PC+1+k，否則PC=PC+1。
cond=4  JR.N  #k; ;若NF=1則PC=PC+1+k，否則PC=PC+1。
cond=5  JR.P  #k; ;若NF=0則PC=PC+1+k，否則PC=PC+1。
cond=6  JR.V  #k; ;若VF=1則PC=PC+1+k，否則PC=PC+1。
cond=7  JR.NV #k; ;若VF=0則PC=PC+1+k，否則PC=PC+1。
cond=8  JR.LE #k; ;若NF=1或ZF=1則PC=PC+1+k，否則PC=PC+1。
cond=9  JR.GT #k; ;若NF=0且ZF=0則PC=PC+1+k，否則PC=PC+1。
cond=10 JR    #k; ;無條件執行PC=PC+1+k。
cond=11 JR.H  #k; ;無條件執行PC=PC+1+k+HX。
```

K值的計算由編譯程式自動產生，使用者不用費心。其範圍是+/-127，所以只能做小幅度的前後跳躍動作。JR指令的細部分解動作說明如下：

```
JR.c #k;    #1.PC++;
clk=2       #2.if (match) PC+=k;          跳到指定位址繼續執行
```

JR指令採用相對位移，所以在編譯成機械碼後，即使整個模組搬移位置仍然能夠正確的執行。編譯程式也可以接受幾種不同的JR寫法，包括：

JR.C	JC			JR.C 指令
JR.NC	JNC			JR.NC指令
JR.Z	JZ	JR.EQ	JEQ	JR.Z 指令
JR.NZ	JNZ	JR.NE	JNE	JR.NZ指令
JR.N	JN	JR.LT	JLT	JR.N 指令
JR.P	JP	JR.GE	JGE	JR.P 指令
JR.V	JV			JR.V 指令
JR.NV	JNV			JR.NV指令
		JR.LE	JLE	JR.LE指令
		JR.GT	JGT	JR.GT指令
JR.H	JH			JR.H指令

LOOP指令

LOOP指令是執行多指令迴圈的動作。其機械碼定義為

	15 ~ 9	8 ~ 0	
LOOP #k	000-1110	k=+/-255	K0 類群

LOOP指令的細部分解動作說明如下：

```
LOOP #k; #1. PC++;                          PC指到下一位址
clk=2    #2. LC++;                          LC值遞增
         #3. if ((--LP)>0) LC++; PC+=k;     PC指到迴圈的相對位址
             else          LC=0;            將LC值還原成0
```

因為LOOP指令是以相對位移的方式執行。所以在編譯成機械碼後，即使整個模組搬移位置仍然能夠正確的執行。

在組合語言的編譯上，WR.LP指令的下一行指令將會自動視為多指令迴圈的起點。所以LOOP #k指令可以簡化成LOOP指令，使用者不用設定k值，而由編譯器自動換算。例如：

```
    WR.LP #4;        //設定迴圈要執行四次
    .....            //真正的迴圈執行區域
    LOOP;            //迴圈結束點，k值不用特別定義，由編譯程式自動計算
```

MAX指令

MAX指令是比較運算的動作，其機械碼定義為

	15 13	12 9	8 4	0	
MAX #k	000	func=4	k=+/-255		k0類群
MAX #k,bit	001	func=4	bit=0~15	k=0~31	k1類群
MAX r,bit	010	func=4	bit=0~15	reg=0~31	k2類群
MAX r,#k	011	func=4	k=0~15	reg=0~31	k3類群
MAX (r)	100	func=4	mode	reg=0~31	k4類群

MAX指令中,除了運算動作改成(AX=max(AX,HX))之外,其他所有定義和執行細節都和ADC指令相同。

MIN指令

MIN指令是比較運算的動作，其機械碼定義為

	15 13	12 9	8 4	0	
MIN #k	000	func=5	k=+/-255		k0類群
MIN #k,bit	001	func=5	bit=0~15	k=0~31	k1類群
MIN r,bit	010	func=5	bit=0~15	reg=0~31	k2類群
MIN r,#k	011	func=5	k=0~15	reg=0~31	k3類群
MIN (r)	100	func=5	mode	reg=0~31	k4類群

MIN指令中,除了運算動作改成(AX=min(AX,HX))之外,其他所有定義和執行細節都和ADC指令相同。

MUL指令

MUL指令是乘法運算的動作，其機械碼定義為

	15 13	12 9	8 4	0	
MUL #k	000	func=3	k=+/-255		k0類群
MUL #k,bit	001	func=3	bit=0~15	k=0~31	k1類群
MUL r,#k	011	func=3	k=0~15	reg=0~31	k3類群
MUL (r)	100	func=3	mode	reg=0~31	k4類群

MUL指令中，除了運算動作改成(HXAX=AX*HX)，並且沒有K2類群的MUL r,bit動作，其他定義和執行細節都和ADC指令相同。

MUL本身是(16-bit * 16-bit = 32-bit)的精度，所以其輸出必須用HX/AX兩組暫存來器儲存，其中HX暫存器儲存高位，而AX暫存器儲存低位。若想取得16-bit的輸出，可在MUL指令之後立刻用SFR指令作移位動作，再從A暫存器中取出移位後的16-bit結果。

NEG指令

NEG指令是變更正負號的動作，其機械碼定義為

	15 13	12 9	8 4	0	
NEG r	011	func=11	0000	reg=0~31	k3類群
NEG	100	func=11	mode=4	reg=0~31	k4類群

和ABS指令一樣，NEG指令有兩種模式，就是

NEG; ;AX=-AX 的正負互換動作。

NEG r; ; r=-r 的正負互換動作。

每個指令的細部分解動作說明如下：

NEG;	#1.PC++;	
clk=1	#2.AX=-AX	運算動作，無重複控制

NEG reg;	#1.PC++; DP=MP+LC+reg+0x800	LC為迴圈參數，預設=0
clk=4	#2.AX=(DP)	讀入動作
	#3.AX=-AX	運算動作
	#4.(DP)=AX	寫出動作，無重複控制

NOP指令

NOP指令不做任何處理，只是佔用一個機械碼空間而已。其機械碼定義為

	15　13	12　9	8　4	0	
NOP	101	func=3	0000	00000	N5類群

NOP指令的細部分解動作說明如下：

```
NOP;        #1.PC++;
clk=1+n     #2.DP++; if ((--RPT)>0) loop#2 此項配合重複指令控制
```

NOP的用途有兩種，一種是產生額外的延遲，用以配合硬體電路的時序需求。另一種用途是產生額外的機械碼，用以控制程式的長度。下面的範例是配合JR.H指令的動作：

```
     RD idx; MUL #5; WR.H; JR.H tab; //跳到tab+(idx*5)的位址繼續執行
tab: ***; ***; ***; ***; JR tend;    //若idx=0則執行此部分，長度固定爲5-word
     ***; ***; NOP; NOP; JR tend;    //若idx=1則執行此部分，長度不足時以NOP補齊
     .....
     ***; ***; ***; ***; JR tend;    //若idx=N則執行此部分，長度固定爲5-word
tend:
```

OR指令

OR指令是邏輯運算的動作，其機械碼定義為

	15–13	12–9	8–4	4–0	
OR #k	000	func=1	k=+/-255		k0類群
OR #k,bit	001	func=1	bit=0~15	k=0~31	k1類群
OR r,bit	010	func=1	bit=0~15	reg=0~31	k2類群
OR r,#k	011	func=1	k=0~15	reg=0~31	k3類群
OR (r)	100	func=1	mode	reg=0~31	k4類群

OR指令中，除了運算動作改成(AX=AX.or.HX)之外，其他所有定義和執行細節都和ADC指令相同。

POP指令

POP指令將資料從堆疊器中取出，其機械碼定義為

	15 13	12 9	8 4	0	
POP (r)	100	func=14	mode	reg=0~31	k4類群

其中mode=0~15，而根據mode的不同，包括了下面的多種模式：

```
mode=11 POP.H;   ;HX =(--SP)的堆疊器取出動作。
mode=12 POP.A;   ;AX =(--SP)的堆疊器取出動作。
mode=13 POP.F;   ;FX =(--SP)的堆疊器取出動作。
mode=14 POP  r;  ; r =(--SP)的堆疊器取出動作，r屬於模組變數區。
mode=15 POP (r); ;(r)=(--SP)的堆疊器取出動作，r屬於模組變數區。
```

每個指令的細部分解動作說明如下：

```
POP.H;    #1.PC++
clk=1     #2.HX=(--SP)                  從堆疊器中取出資料，無重複控制
```

```
POP.A;    #1.PC++
clk=1     #2.AX=(--SP)                  從堆疊器中取出資料，無重複控制
```

```
POP.F;    #1.PC+
clk=1     #2.FX=(--SP)                  從堆疊器中取出資料，無重複控制
```

```
POP  reg;  #1.PC++; DP=MP+LC+reg+0x800     LC爲迴圈參數，預設=0
clk=2+n    #2.(DP)=(--SP)                  從堆疊器中取出資料
           #3.DP++; if ((--RPT)>0) loop#2  此項配合重複指令控制
```

```
POP (reg); #1.PC++; DP=MP+LC+reg+0x800     LC爲迴圈參數，預設=0
clk=3+n    #2.DP=(DP)                      間接讀寫動作
           #3.(DP)=(--SP)                  從堆疊器中取出資料
           #4.DP++; if ((--RPT)>0) loop#3  此項配合重複指令控制
```

PRT指令

PRT指令是保護整數運算的過溢(Overflow)現象。其機械碼定義為

	15　13	12　9	8　4	0	
PRT.0	101	func=6	0000	00000	N5類群
PRT.F	101	func=7	0000	00000	N5類群

PRT指令的細部分解動作說明如下：

```
PRT.0;  #1.PC++
clk=1   #2.if      (VF=1)              then AX=0;          過溢保護動作
```

```
PRT.F;  #1.PC++
clk=1   #2.if       (VF=1 and NF=1) then AX=0x7fff      過溢保護動作
              else if (VF=1 and NF=0) then AX=0x8001      過溢保護動作
```

當整數計算出現過溢現象時，正數會變成負數，而負數會變成正數。為了降低系統錯誤的影響，最安全的做法就是將過溢現象轉化成飽和(Saturation)現象。對於整數的飽和值，正整數是0x7fff，而負整數是0x8001。這時就可以用PRT.F指令來作保護。

對於32-bit的整數運算，正整數的飽和值為0x7fff0000，而負整數的飽和值為0x80010000。這時就可用PRT.F指令來保護高位的運算結果，而用PRT.0指令來保護低位的運算結果。

PUSH指令

PUSH指令將資料儲存於堆疊器中，其機械碼定義為

	15 13	12 9	8 4	0	
PUSH (r)	100	func=14	mode	reg=0~31	k4類群

其中mode=0~15，而根據mode的不同，包括了下面的多種模式：

```
mode=6  PUSH.H;     ;(SP++)=HX 的堆疊器儲存動作。
mode=7  PUSH.A;     ;(SP++)=AX 的堆疊器儲存動作。
mode=8  PUSH.F;     ;(SP++)=FX 的堆疊器儲存動作。
mode=9  PUSH  r;    ;(SP++)= r 的堆疊器儲存動作，r屬於模組變數區。
mode=10 PUSH (r);   ;(SP++)=(r)的堆疊器儲存動作，r屬於模組變數區。
```

而每個指令的細部分解動作說明如下：

PUSH.H;	#1.PC++	
clk=1	#2.(SP++)=HX	儲存至堆疊器中，無重複控制

PUSH.A;	#1.PC++	
clk=1	#2.(SP++)=AX	儲存至堆疊器中，無重複控制

PUSH.F;	#1.PC++	
clk=1	#2.(SP++)=FX	儲存至堆疊器中，無重複控制

PUSH reg;	#1.PC++; DP=MP+LC+reg+0x800	LC爲迴圈參數，預設=0
clk=1+n	#2.(SP++)=(DP)	儲存至堆疊器中
	#3.DP++; if ((--RPT)>0) loop#2	此項配合重複指令控制

PUSH (reg);	#1.PC++; DP=MP+LC+reg+0x800	LC爲迴圈參數，預設=0
clk=2+n	#2.DP=(DP)	間接讀寫動作
	#3.(SP++)=(DP)	儲存至堆疊器中
	#4.DP++; if ((--RPT)>0) loop#3	此項配合重複指令控制

RD指令

RD指令是將資料讀入AX暫存器的動作，其機械碼定義包括

	15 13	12 9	8 4	0	
RD #k	000	func=15	k=+/-255		k0類群
RD #k,bit	001	func=15	bit=0~15	k=0~31	k1類群
RD r,bit	010	func=15	bit=0~15	reg=0~31	k2類群
RD (r)	100	func=15	mode	reg=0~31	k4類群

其中mode=0~15，而根據mode的不同，包括了下面的多種模式：

```
mode=0  RD.0  r;    ;AX= r 的直接讀取動作，r屬於公用變數區。
mode=1  RD.0 (r);   ;AX=(r)的間接讀取動作，r屬於公用變數區。
mode=2  RD    r;    ;AX= r 的直接讀取動作，r屬於模組變數區。
mode=3  RD   (r);   ;AX=(r)的間接讀取動作，r屬於模組變數區。
mode=4  RD.H;       ;AX= HX 的暫存器讀取動作。
mode=5  RD.M;       ;AX=(HX)的表格讀取動作，可作爲查表指令。
mode=6  RD.S  r;    ;先執行PUSH A的儲存動作後，再執行RD  r 指令。
mode=7  RD.S (r);   ;先執行PUSH A的儲存動作後，再執行RD (r)指令。
mode=8  RD.X  r;    ;AX<=> r 的置換動作。
mode=9  RD.X (r);   ;AX<=>(r)的置換動作。
mode=10 RD.X;       ;AX<=>HX 的置換動作。
```

而每個指令的細部分解動作說明如下：

```
RD.0   reg;   #1.PC++; DP=reg               取得公用變數區的位址
clk=1         #2.AX=(DP)                    讀取資料，無重複控制
```

```
RD.0 (reg);   #1.PC++; DP=reg               取得公用變數區的位址
clk=2         #2.DP=(DP)                    間接讀寫動作
              #3.AX=(DP)                    讀取資料，無重複控制
```

```
RD     reg;   #1.PC++; DP=MP+LC+reg+0x800   LC爲迴圈參數，預設=0
clk=1         #2.AX=(DP)                    讀取資料，無重複控制
```

```
RD   (reg);   #1.PC++; DP=MP+LC+reg+0x800   LC爲迴圈參數，預設=0
clk=2         #2.DP=(DP)                    間接讀寫動作
              #3.AX=(DP)                    讀取資料，無重複控制
```

```
RD.H;         #1.PC++
clk=1         #2.AX=HX                      讀取資料，無重複控制
```

```
RD.M;         #1.PC++;
clk=2         #2.DP=HX++;                      以HX值設定表格位址
              #3.AX=(DP);                      讀取資料，無重複控制
```

```
RD.S  reg;    #1.PC++; DP=MP+LC+reg+0x800;     LC爲迴圈參數，預設=0
clk=2+2n      #2.(SP++)=AX;                    將AX暫存器儲存至堆疊區中
              #3.AX=(DP);                      讀取資料
              #4.DP++; if ((--RPT)>0) loop#2   此項配合重複指令控制
```

```
RD.S (reg);   #1.PC++; DP=MP+LC+reg+0x800      LC爲迴圈參數，預設=0
clk=3+2n      #2.DP=(DP)                       間接讀寫動作
              #3.(SP++)=AX                     將AX暫存器儲存至堆疊區中
              #4.AX=(DP)                       讀取資料
              #5.DP++; if ((--RPT)>0) loop#3   此項配合重複指令控制
```

```
RD.X reg;     #1.PC++; DP=MP+LC+reg+0x800      LC爲迴圈參數，預設=0
clk=3+3n      #2.HX=AX; AX=(DP)                將AX儲存後再讀取資料
              #3.(DP)=HX                       將HX暫存器儲存回記憶體中
              #4.DP++; if ((--RPT)>0) loop#2   此項配合重複指令控制
```

```
RD.X (reg);   #1.PC++; DP=MP+LC+reg+0x800      LC爲迴圈參數，預設=0
clk=4+3n      #2.DP=(DP)                       間接讀寫動作
              #3.HX=AX; AX=(DP)                將AX儲存後再讀取資料
              #4.(DP)=HX                       將HX暫存器儲存回記憶體中
              #5.DP++; if ((--RPT)>0) loop#3   此項配合重複指令控制
```

```
RD.X;         #1.PC++;
clk=1         #2.AX<=>HX                       AX和HX交換，無重複控制
```

```
RD reg,bit;   #1.PC++; DP=MP+LC+reg+0x800      LC爲迴圈參數，預設=0
clk=1         #2.AX=(DP)<<bit                  讀取資料，無重複控制
```

```
RD #k,bit;    #1.PC++
clk=1         #2.AX=k<<bit                     讀取資料，無重複控制
```

```
RD #k;        #1. PC++
clk=1         #2. AX=k                         讀取資料，無重複控制
```

RET指令

RET指令是副程式結束而返回主程式的動作。其機械碼定義為

	15 13	12 9	8 4	0	
RET	101	func=4	0000	00000	N5類群

RET指令的細部分解動作說明如下：

RET; clk=2	#1.PC=(--SP)	取出儲存至堆疊器中的程式位址後返回

RPT指令

RPT指令是重複指令，下一指令將重複k次。其機械碼定義為

	15　　13	12　　　9	8　　　　4　　　　0	
RPT #k	000	func=12	k=+/-255	k0類群

RPT指令的細部分解動作說明如下：

RPT #k;	#1.PC++	
clk=1	#2.RPT=k;	設定重複次數

重複次數k值必須限制在1~255之間。

SBC指令

SBC指令是減法運算的動作，其機械碼定義為

	15 13	12 9	8 4	0	
SBC #k	000	func=9	k=+/-255		k0類群
SBC #k,bit	001	func=9	bit=0~15	k=0~31	k1類群
SBC r,bit	010	func=9	bit=0~15	reg=0~31	k2類群
SBC r,#k	011	func=9	k=0~15	reg=0~31	k3類群
SBC (r)	100	func=9	mode	reg=0~31	k4類群

SBC指令中，除了運算動作改成(AX=AX-HX-CF)之外，其他所有定義和執行細節都和ADC指令相同。

SFL指令

SFL指令是作向左移位的動作，其機械碼定義為

	15–13	12–9	8–4	4–0	
SFL bit	010	func=14	bit=0~15	00000	k2類群

SFL指令只有一種模式，就是

```
SFL bit;    ;AX=AX<<bit的左移動作。
SFL;        ;如果bit省略，則預設bit=1。
```

其中bit=0~15。SFR指令的細部分解動作說明如下：

```
SFL bit;  #1.PC++
clk=1     #2.AX=AX<<bit              左移運算動作，無重複控制
```

SFR指令

SFR指令是作向右移位的動作，其機械碼定義為

	15 13	12 9	8 4	0	
SFR bit	010	func=13	bit=0~15	00000	k2類群

SFR指令只有一種模式，就是

```
SFR bit;    ;AX=HX,AX>>bit的右移動作。
SFR;        ;如果bit省略，則預設bit=1。
```

其中bit=0~15。SFR指令的細部分解動作說明如下：

SFR bit;	#1.PC++	
clk=1	#2.AX=HX,AX>>bit	右移運算動作，無重複控制

因為是向右移位必須特別注意的是：SFR指令是32-bit向右移位後取出16-bit的動作，所以其輸入是32-bit的HX,AX暫存器而不只是16-bit的AX暫存器。對於16-bit資料，必須先配合SGN指令擴張成32-bit後，才能使用SFR指令。

SGN指令

SGN指令是將16-bit暫存器擴張到32-bit暫存器的動作，其機械碼定義為

	15　13	12　9	8　4	4　0	
SGN bit	010	func=12	bit=0~15	00000	k2類群

SGN指令只有一種模式，就是

SGN bit;　;HX,AX=sign(AX,bit)的正負號充填動作。
SGN;　　　;如果bit省略，則預設bit=15。

必須注意的是：SGN指令是少數將計算結果直接存到HX暫存器的指令。

	HX	AX
AX		AX
bit		0000100000000000
	SSSSSSSSSSSSSSSS	SSSSSXXXXXXXXXXX

說明如下：

1. 根據bit值的定義，以AX(bit)爲正負號的依據。
2. 根據正負號的偵測結果，將AX中高於bit值的位元都以正負號補齊。
3. 同樣的，將HX設爲0000或FFFF，也是以正負號補齊。
4. 補位指令執行時，只有NF和ZF兩個旗標更新。

其細部分解動作說明如下：

SGN bit;	#1.PC++;	
clk=1	#2.HX,AX=sign(AX,bit)	正負號充填動作，無重複動作

SUB指令

SUB指令是減法運算的動作，其機械碼定義包括：

	15 13	12 9	8 4	0	
SUB #k	000	func=8	k=+/-255		k0類群
SUB #k,bit	001	func=8	bit=0~15	k=0~31	k1類群
SUB r,bit	010	func=8	bit=0~15	reg=0~31	k2類群
SUB r,#k	011	func=8	k=0~15	reg=0~31	k3類群
SUB (r)	100	func=8	mode	reg=0~31	k4類群

SUB指令中，除了運算動作改成(AX=AX-HX)之外，其他所有定義和執行細節都和ADC指令相同。

WR指令

WR指令是設定記憶體的動作，其機械碼定義包括：

	15~13	12~9	8~4	3~0	
WR.LP #k	000	func=13	k=+/-255		k0類群
WR r,#k	011	func=15	k=0~15	reg=0~31	k3類群
WR (r)	100	func=14	mode	reg=0~31	k4類群
WR.MP #k	1111	k=0~4095			J7類群

這四類指令的定義完全不同，首先說明K4類群。其中mode=0~15，而根據mode的不同，包括了下面的多種模式：

```
mode=0 WR.0   r;   ; r =AX的直接儲存動作，r屬於公用變數區。
mode=1 WR.0 (r);  ;(r)=AX的間接儲存動作，r屬於公用變數區。
mode=2 WR     r;   ; r =AX的直接儲存動作，r屬於模組變數區。
mode=3 WR   (r);  ;(r)=AX的間接儲存動作，r屬於模組變數區。
mode=4 WR.H;       ; HX =AX的暫存器儲存動作。
mode=5 WR.M;       ;(HX)=AX的表格儲存動作，可作為查表指令。
```

而每個模式的細部分解動作說明如下：

```
WR.0  reg;  #1.PC++; DP=reg                取得公用變數區的位址
clk=2+n     #2.(DP)=AX                     儲存動作
            #3.DP++; if ((--RPT)>0) loop#2 此項配合重複指令控制
```

```
WR.0 (reg); #1.PC++; DP=reg                取得公用變數區的位址
clk=3+n     #2.DP=(DP)                     間接讀寫動作
            #3.(DP)=AX                     儲存動作
            #4.DP++; if ((--RPT)>0) loop#3 此項配合重複指令控制
```

```
WR    reg;  #1.PC++; DP=MP+LC+reg+0x800;   LC為迴圈計數，預設=0
clk=2+n     #2.(DP)=AX                     儲存動作
            #3.DP++; if ((--RPT)>0) loop#2 此項配合重複指令控制
```

```
WR   (reg); #1.PC++; DP=MP+LC+reg+0x800;   LC為迴圈計數，預設=0
clk=3+n     #2.DP=(DP)                     間接讀寫動作
            #3.(DP)=AX                     儲存動作
            #4.DP++; if ((--RPT)>0) loop#3 此項配合重複指令控制
```

```
WR.H;     #1.PC++
clk=1     #2.HX=AX                         無重複控制
```

```
WR.M;     #1.PC++;
clk=3+n   #2.DP=HX++;                        取得表格位址
          #3.(DP)=AX;                        儲存動作
          #4.DP++; if ((--RPT)>0) loop#3     此項配合重複指令控制
```

K3類群的(WR r,#k)指令是直接設定暫存器的內容，分解動作如下：

```
WR reg,#k;  #1.PC++ DP=MP+LC+reg+0x800;    LC爲迴圈計數，預設=0
clk=2+n     #2.(DP)=k;                     儲存動作
            #3.DP++; if ((--RPT)>0) loop#2 此項配合重複指令控制
```

K0類群的(WR.LP #k)指令是設定多重迴圈的次數，分解動作如下：

```
WR.LP #k;   #1.PC++
clk=1       #2.LP=k; LC=0                  設定多重迴圈的次數
```

其中重複次數k值必須限制在1~255之間。

J7類群的(WR.MP #k)指令是設定模組變數區的起使位址，分解動作如下：

```
WR.MP #k;   #1.PC++
clk=1       #2.MP=k;                       設定模組變數區的起點
```

XOR指令

XOR指令是邏輯運算的動作，其機械碼定義包括

	15　13	12　9	8　4	0	
XOR #k	000	func=2	k=+/-255		k0類群
XOR #k,bit	001	func=2	bit=0~15	k=0~31	k1類群
XOR r,bit	010	func=2	bit=0~15	reg=0~31	k2類群
XOR r,#k	011	func=2	k=0~15	reg=0~31	k3類群
XOR (r)	100	func=2	mode	reg=0~31	k4類群

XOR指令中，除了運算動作改成(AX=AX.xor.HX)之外，其他的所有定義和執行細節都和ADC指令相同。

5.模組程式設計

模組程式是由組合語言撰寫的程式。要設計程式之前，必須先瞭解模組程式的基本架構，包括控制程序、副程式、跳躍指令、變數、輸入輸出點、表格和硬體界面等等，以便正確的規劃模組程式。

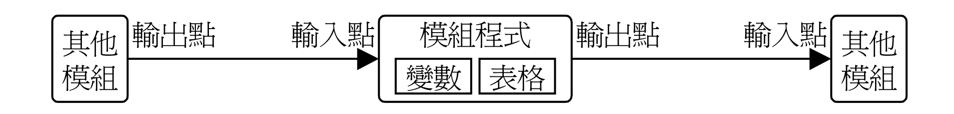
在編譯程序方面，則包括程式撰寫和編譯動作，並應確實瞭解編譯後的檔案說明以備偵錯使用。

在本章的最後，將以數位控制器、波形產生器和資料記錄器為例，對設計重點作提示，以提供程式設計者做參考。

5.1 程式架構

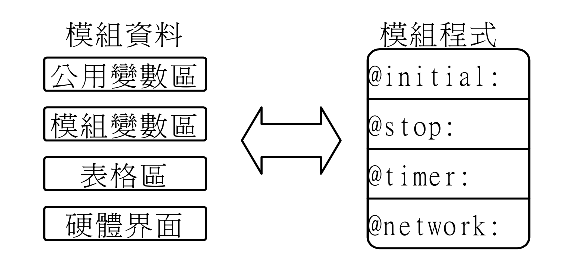
模組架構

就結構而言，模組程式的系統方塊圖如下，其中包括：



1.模組程式允許多組輸入點，可以用來連接其他模組的輸出點。
2.模組程式允許多組輸出點，可以用來連接其他模組的輸入點。
3.模組程式中可以擁有內部變數和表格，用以執行指定的計算或是控制工作。

撰寫時模組程式是完全獨立的，和其他模組無關。就程式結構而言，模組程式包括了程式和資料兩個部分。其中資料分成公用變數、模組變數、表格和硬體界面四部分：



1.公用變數是所有模組所共享的變數區，在模組中依需求來使用。
2.模組變數是模組所獨佔的變數區，各模組間互不干涉。
3.表格是各個模組可以佔用的資料區，但必須以間接模式讀寫。
4.硬體界面是具有絕對位址的記憶體，可以用間接模式來讀寫。

程式區中包括了四個副程式，由系統程式負責呼叫。四個副程式包括：

@initial程序：控制系統啓動時執行一次。
@stop程序：　控制系統關閉時執行一次。
@timer程序：　控制系統中的週期性控制程式，每隔指定的抽樣週期執行一次。
@network程序：控制系統中的不定期控制程式，由外界的CPU啓動，負責和其他CPU的控制同步。例如當外界的CPU負責網路通訊時，每次當有網路資料更新動作時，啓動此程序一次。

檔案結構

模組程式是以文書檔的型式存在，副檔名固定為".ASM"，文法架構如下：

```
//說明文件
標籤：                          //單行中只有標籤
       指令;                    //單行中只有指令
標籤： 指令;                    //標籤與指令可以寫在同一行
       指令; 指令; 指令;        //多項指令也可以寫在同一行
```

其中的定義包括：

1.組合語言中的指令固定以英文撰寫，大小寫一視同仁，沒有區別。但我們通常用大寫做指令而小寫做變數，以方便閱讀。
2.指令以單行爲準，超過一行的部分將視爲下一個新指令的開始。
3.以"//"區分說明文件，從"//"開始到單行結束間的字串都視爲說明文件，編譯器中將不做處理。在說明文件中則允許中文的撰寫方式。
4.多項指令可以合併在一行中撰寫。編譯器只要檢查到";"符號就認定指令已經結束，而將";"之後的字串視爲下一個新指令的開始。
5.程式中可以加標籤(Label)，以指定標籤所在的程式位址。標籤以":"爲結尾，而在":"之後的字串視爲下一個新指令的開始。
6.在指令和標籤中以@開頭的字串都是編譯指令，用以指示編譯器做特殊用途或定義，而不會翻譯成機械碼。

而最基本的程式架構如下：

```
@global    .....;             //定義公用變數
@local     .....;             //定義模組變數
@data      .....;             //定義模組變數區的長度
@initial: .....              //@initial副程式，設定啓始値
           RET;               //以RET指令結束副程式
@stop:     ....              //@stop副程式，設定安全的停止狀態
           RET;               //以RET指令結束副程式
@timer:    ....              //@timer副程式，週期性的即時控制程式
           RET;               //以RET指令結束副程式
@network: ....               //@network副程式，不定期的即時控制程式
           RET;               //以RET指令結束副程式
```

其中包括：

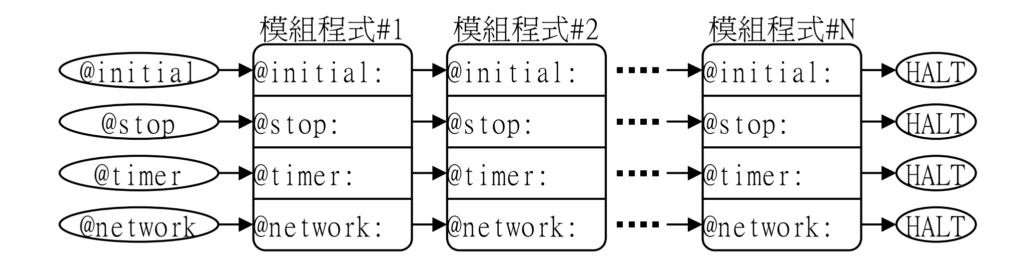
1.以@global指令定義公用變數，每一個公用變數都要單獨定義。
2.以@local 指令定義模組變數，每一個模組變數都要單獨定義。
3.以@data　指令定義模組變數區的長度，最大爲32-word。
4.以@initial標籤定義@initial程序的開始，而以RET指令結束該程序。
5.以@stop　 標籤定義@stop　 程序的開始，而以RET指令結束該程序。
6.以@timer　標籤定義@timer　程序的開始，而以RET指令結束該程序。

7.以@network標籤定義@network程序的開始，而以RET指令結束該程序。

下面則針對各個部分作詳細說明。

程序的撰寫

每個程序都是副程式結構，而以RET指令做結束。當系統程式連結多個模組時，就會自動的依序呼叫各模組中的相對程序。圖示如下：



以@timer程序為例，當DSP核心週期性的產生@timer需求時，系統程式就會先執行模組程式#1中的@timer程序，接著再依序執行其他模組程式中的@timer程序。最後在系統程式中再以HALT指令結束，將控制權交還給硬體，準備下一個程序的觸發動作。

如果模組中並不需要處理某項程序時，在程式中就不要定義該項程序。當系統程式執行時會自動的跳過這個模組而不處理。例如程式中只有@initial、@stop和@timer三項時，系統在執行@network程序時就不會呼叫這個模組，但是其他程序仍然會正常工作。

副程式的呼叫

如果有需要，各個程序中還可以再呼叫其他副程式。例如：

```
@timer:    ....               //@timer程序，週期性的即時控制程式
           CALL  fun1;          //程序中還可以呼叫副程式
           RET;                 //以RET指令結束程序
fun1:      CALL  fun2;          //副程式中還可以再呼叫其他副程式
           .....;  RET;         //以RET指令結束副程式
fun2:      .....;  RET;         //以RET指令結束副程式
```

副程式的名稱以標籤來定義，標籤所在的位址就是副程式的啟始位置，而以RET指令來結束副程式。呼叫副程式是CALL指令來完成，因為CALL指令是以相對位移的方式來計算副程式的位址，所以在編譯完成後，即使模組搬移到其他位置，還是能夠正確無誤的執行。

在模組程式的架構下，所有的標籤名稱都是局部的。換句話說，不同的模組程式可以用同一個標籤名稱，但是兩者之間卻完全獨立。但是相對的，副程式只有在同一模組內有

效，不同模組之間不能共享副程式。

另一方面，副程式允許階層式的呼叫。換句話說，副程式中還可以呼叫其他副程式，次數不限。副程式的程式長度也無限制，只要能放入程式區即可。

跳躍指令的撰寫

跳躍指令通常搭配標籤來進行，例如：

```
        SUB cmd,#1; JZ test1;      //如果ZF=1則跳到test1去執行，否則繼續
         .....;      JR test2;      //直接跳到test2去執行
test1:   .....;                     //test1的分支程式
test2:   .....;                     //test2的分支程式
```

其中標籤的位置就是跳躍指令可以指定的地址。由於跳躍指令是以相對位移來定義要執行的新位址，其相對位移的範圍為+/-127，所以跳躍的範圍也就被限制在JR指令的前後一小段程式中了。

公用變數的定義

公用變數區的長度固定為32-word，由所有模組所共享，所以不能隨便的更改定義，必須在系統設計前仔細規劃後才能定案。最常用的公用變數包括：

nTMR(R1~0)：@timer程序計數，控制啓動時預設爲0，每次@timer程序都會由系統程式自動加一，用以鑑別目前的系統時間。nTMR爲32位元變數，以32KHz的@timer週期而言，約可計時36小時，再繼續循環計數。
nNET(R2)　：@network程序計數，控制啓動時預設爲0，每次@network程序都會由系統程式自動加一，可做控制上的參考。nNET爲16位元，可循環計數。
nALARM(R3)：通用警報狀態，控制啓動時預設爲0。nALARM爲16位元，其個別意義由程式設計者自行定義。

至於其他的公用變數，則可由使用者自定義。公用變數的指令定義如下：

```
@global  <name> <reg>;                    //定義公用變數
```

其中：

1. @global爲公用變數的定義指令。
2. <name>爲以英文字母開頭的任意名稱，長度限制爲32字，不得和模組中的任何其他名稱重複。
3. <reg>爲R0~R31的固定名稱，代表公用變數區的32個變數。

一旦以@global指令來定義公用變數後，該名稱即可在程式中使用。例如：

```
@global nTMR  R0;                //定義公用變數(nTMR=R0)
....    RD.0 nTMR;                   //相當於 RD.0 R0指令
.....   RD.0 R0;                     //不論有無@global定義，R0~R31都可使用
```

但是即使沒有用@global指令定義公用變數，仍然可以用R0~R31來指定公用變數的位置。換句話說，@global指令相當於C語言中的#define指令，使程式的讀寫更加容易而已，並沒有實質的機械碼編譯動作。

模組變數的定義

模組程式中可以使用32-word的模組變數，模組變數本身可獨立規劃或使用，和其他模組無關。指令定義如下：

```
@local <name> <reg> DP;                      //定義模組變數，DP參數也可以省略
@local <name> <reg> DP <init>;               //定義模組變數和其初值
@local <name> <reg> DP <init> <unit> <bias>; //定義模組變數、初值和顯示參數
@data  <length>;                               //定義模組變數區的長度
```

其中：

1. @local為模組變數的定義指令，DP參數通常可以省略。
2. <name>為以英文字母開頭的任意名稱，長度限制為32字，不得和模組中的任何其他名稱重複。
3. <reg>為R0~R31的固定名稱，代表模組變數區的32個變數。
4. 可以用<init>參數設定初值，初值是+/-32767之間的整數，當程式下載時會自動的設定變數的初值。若無特別指定，初值的預設值為0。
5. 一般而言，模組變數是16位元的整數，但是常常會對應成浮點數值顯示，這時就可以用<unit>和<bias>參數來設定浮點的對應公式。公式如下：

   F = (I-bias)/unit，而

   I = F*unit + bias

   其中I為16位元整數，而F為其對應的浮點數值。
6. 若無特別定義，則bias=0而unit=1。換句話說，浮點數和整數的數值一致。
7. 以@data指令定義模組變數區的長度，可選擇0~32的值。若無特別指定，則預設值為0，代表用不到模組變數區。

一旦以@local指令來定義模組變數後，該名稱即可在程式中使用。但是即使沒有用@local指令來定義模組變數，仍然可以用R0~R31來指定模組變數的位置。換句話說，@local指令相當於C語言中的#define指令，使程式的讀寫更加容易而已，並沒有實質的機械碼編譯動作。

模組的輸入點

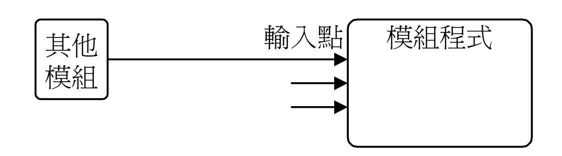
模組的輸入點和其他模組變數一樣的用@local指令定義，但是其參數稍微不同。定義如下：

```
@local <name> <reg> IP;            //定義模組輸入點
@local <name> <reg> IP <pnt>;      //定義模組輸入點和其預設指標
```

其中：

1. IP是特定參數，用以指示這個模組變數是當成輸入點使用。
2. 通常輸入點的變數內容就是指標，指向另一個模組的輸出點，而以RD (r)指令的方式讀入。
3. 以<pnt>參數做輸入點指標的預設工作。

在模組程式設計時，並不能事先知道將來的系統連線關係，所以輸入點的指標無法預設。而是在系統程式指定其連線關係後，才能將正確的指標填入。所以輸入點預設指標的用途非常特殊，說明如下：



1. 假設模組中有一個或多個的輸入點，理論上這些輸入點都是指標，必須由系統程式填入。
2. 但是常常在系統程式設計時，並不需要用到所有的輸入點。這時除非輸入點有某個預設指標，不然整個模組運算就會完全失控。

換句話說，輸入點的預設指標只有一個用途，就是做選擇性的輸入點，萬一系統程式放棄使用時，還能正確的運用這個模組程式。

下面用個好例子來做說明：

1. 假設我們要做一個正弦波產生器，可以控制其頻率、振幅和相位。
2. 如果將頻率、振幅和相位當作三個參數(也就是模組變數)，只要改變這三個變數，就能改變正弦波的特值。
3. 但是只要稍加改變，就能使這個模組的功能大幅提升。例如將頻率參數改變成頻率輸入點，那麼單純的正弦波產生器立刻就升格成調頻模組，可接受其他模組的控制產生調頻波。
4. 依此類推，同一個模組就可變身爲調頻模組、調幅模組和調相模組。
5. 所以同樣的頻率、振幅和相位，可以做爲一般的變數或是做爲輸入點，必須要由系統程式做最後決定。

因此，我們在模組程式中做了下面的設定：

```
@local  frq  R1 IP kfrq;              //定義輸入點frq，而其預設指標為kfrq
@local  kfrq R2    1000;              //定義kfrq的預設值為1000
...     RD (frq);                     //以標準的輸入點方式讀入frq命令
```

如果在系統程式中對輸入點frq沒有連線，因為frq的預設值指到kfrq，所以會以1000的預設值作為固定頻率，也可以經由更改kfrq值而更改頻率，這只是傳統上可以設定頻率的正弦波產生器。

但是如果在系統程式中對輸入點frq做了連線，那麼真正的連線指標就會蓋掉輸入點frq中的預設值，這時模組的頻率命令就完全由其他模組控制，而成為高功能的調頻模組了。

模組的輸出點

模組的輸出點和其他模組變數一樣的用@local指令定義，但是其參數稍微不同。定義如下：

```
@local <name> <reg> OP;                        //定義模組輸出點
@local <name> <reg> OP <init>;                 //定義模組輸出點和其初值
@local <name> <reg> OP <init> <unit> <bias>;   //定義模組輸出點、初值和顯示參數
```

其中：

1. 和一般的模組變數相似，只是增加了OP的特定參數而已。
2. 實際上，輸出點的作用和一般的模組變數完全相同。所以要另外定義輸出點，只是要將該模組的連線功能說明的更清楚而已，不論在程式撰寫或是實際控制上都完全沒有差別。

表格的運用

表格的使用是藉著記憶體的間接讀寫來完成，例如：

RD (r)指令：　將reg的內容當成地址，找到表格的位置而讀取之。
WR (r)指令：　將reg的內容當成地址，找到表格的位置而儲存之。

換句話說，在模組變數區中一定要有變數指向表格的位址。因此表格的定義方式如下：

```
#local <name>  <reg> TP;    //定義模組變數，並以TP指定為表格指標
@table <name>  <length>; //定義表格的名稱、長度，其中表格名稱必須和變數名稱一致
     { ..... };             //和初值
     { ..... };             //初值允許用多行撰寫
```

其中：

1.先以@local指令定義模組變數，用以指向表格的位址。
2.再以@table指令定義表格的長度，其中@table指令和@local指令中的名稱必須一致。編譯器能將兩者對應後，會自動將表格的實際位址存入模組變數中。
3.表格中可以定義初值，當初值個數不足時，會自動以0來填滿表格。

在模組程式編譯時會紀錄表格的名稱和資料,當系統組合多個模組後會自動找到空間放置表格，並將表格的位址設定在對映的模組變數中。因此在程式執行時，RD(r)等間接讀寫指令將正確找到表格的真正位址。

硬體界面的管理

硬體界面的讀寫程序和一般的模組變數類似，舉例說明如下：

```
@local   adc1 R0 0xc8;              //定義硬體輸入界面adc0，預設位址爲0xc8
@local   dac1 R2 0xd8;              //定義硬體輸出界面dac0，預設位址爲0xd8
...      RD (adc1);                     //以間接處理讀入硬體輸入界面
...      WR (dac1);                     //以間接處理寫入硬體輸出界面
```

其中：
1.以一般模組變數的方式定義輸入輸出阜，並以其實際位址設定爲初值。
2.以間接讀入方式處理輸入阜，例如上例中的RD (adc0)指令。
3.以間接寫入方式處理輸出阜，例如上例中的WR (dac0)指令。

由於在FPGA中的硬體界面可依需求更換，所以要執行程式前必須先確認FPGA版本，並要查清楚該硬體界面的實際位址和功能定義。

5.2 編譯程序

撰寫模組程式

模組程式的標準格式如下：

```
@global <name>  <reg>;          //定義公用變數
@local  <name>  <reg>  <type>;  //定義模組變數
@data           <length>;       //定義模組變數區的長度
@table  <name>  <length>;       //定義表格的名稱和長度
@initial: .... RET;            //@initial程序，設定啓始狀態
@stop:    .... RET;            //@stop   程序，設定安全的停止狀態
@timer:   .... RET;            //@timer  程序，週期性的即時控制程式
@scanner: .... RET;            //@scanner程序，不重要的掃瞄控制程式
```

其中的重點項目包括：

1. 定義所有的變數，包括公用變數和模組變數，也包括輸入和輸出點。
2. 定義模組變數區及表格的長度和初值。
3. 以組合語言撰寫@initial、@stop、@timer和@scan四個程序。
4. @initial程序在DSP程式啓動時執行一次，在此設定各參數所需的初值。
5. @stop程序在DSP程式結束時執行一次，在此將硬體設定成安全停止狀態。
6. @timer程序是每隔固定週期所執行的即時控制程式，是主要的控制程式。
7. @scanner程序是不重要的掃瞄控制程式，利用DSP的空餘時間執行。
8. 所有的程序都以RET指令結束。程序中還可呼叫模組中的其他副程式。
9. 在模組中如果某些程序不需執行，可以將該程序整段刪除。

檔案說明

模組程式所編譯出的檔案名稱為<module>.mod，其標準格式如下：

```
@GLOBAL <length>;                                         //定義公用變數區
        <name> <reg> <unit> <bias>;                       //定義公用變數
@LOCAL  <length>;                                         //定義模組變數區
        <name> <reg> <init> <table> <type> <unit> <bias>; //定以模組變數
@INITIAL  <offset>, clk=<n>;        //定義@initial程序的相對位址和所需時間
@STOP     <offset>, clk=<n>;        //定義@stop   程序的相對位址和所需時間
@TIMER    <offset>, clk=<n>;        //定義@timer  程序的相對位址和所需時間
@SCANNER  <offset>, clk=<n>;        //定義@scanner程序的相對位址和所需時間
@PROG     <length>;                 //定義程式區的長度
....;                               //定義程式區的每一筆資料
@TABLE    <length>;                 //定義表格區的長度
....;                               //定義表格區的每一筆資料
```

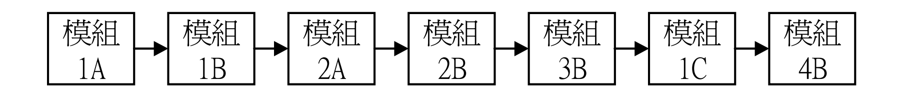
檔案中重點項目說明如下：

1. 紀錄模組名稱，以確認檔案格式。
2. 紀錄@timer副程式的執行時間。
3. 紀錄所有的變數定義，包括公用變數區和模組變數區，也包括輸入輸出點。
4. 紀錄@initial、@stop、@timer和@network副程式的啓始位址，如果缺少某些副程式，則該副程式的啓始位址設為-1。
5. 紀錄程式區的長度和程式碼。
6. 紀錄模組變數區長度和初値。
7. 紀錄所有表格的名稱、長度和初值。

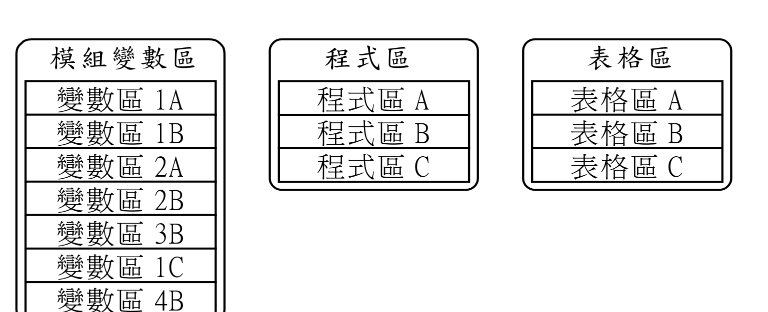
系統記憶體配置

模組程式 A	模組程式 B	模組程式 C
變數區 A	變數區 B	變數區 C
程式區 A	程式區 B	程式區 C
表格區 A	表格區 B	表格區 C

每個模組檔案(.mod)都定義所需要的模組變數區、程式區和表格區的長度及其初值，而系統程式中可以重複的呼叫使用。例如，假設有上面的三個獨立模組。而系統中重複的使用了這三個模組，其中模組A使用兩次、模組B使用4次，而模組C使用一次，如下圖所示。



則在系統程式編譯後，會造成下圖中的記憶體配置，其中



1. 就模組變數區而言，每個模組都有獨立的模組變數區，不論這個模組是否為重複使用。
2. 就程式區而言，每個模組只儲存一組程式。所以當模組重複使用時，就不用重複儲存程式了。
3. 就表格區而言，每個模組只儲存一組表格。所以當模組重複使用時，就不用重複儲存表格了。
4. 換句話說，當模組重複使用時，程式是共用的，表格也是共用的，但是模組變數卻都是獨立的。

如此可以造成最精簡的記憶體配置。但是模組程式撰寫時，一定要記住表格區是共用的這個基本原則，才不會造成資料處理上的錯誤。

6. fPLC 中 C 程式的連結

在fPLC中，C程式可以藉著下面三種步驟作連結：

CALLC程式　：只呼叫一次的C程式
CFUN程式　 ：可以連續呼叫的C程式
C++操作程式：獨立執行的平行作業程式

各個C程式都是藉著公用記憶體來遙控fPLC程式，基本程序包括：

1. 宣告公用記憶體並和fPLC進行連結。
2. 可直接讀寫公用記憶體中的任何資料。
3. 也可藉著外部指令以直接指令的格式對fPLC進行遙控操作。

設計背景

fPLC程式是光纖網路控制中的基本操作環境,所有硬體電路的操作和控制都必須在fPLC環境中來發動。fPLC環境中的操控動作，從最基礎到最高級依序可分成下述幾等：

第一級的操控動作(直接指令控制)：

1.以直接指令下達操控動作。
2.最常用直接指令包括Set/Dump/Move/Fill等等。
3.直接指令動作相當於最基礎的手動控制。

第二級的操控動作(PLC檔控制)：

1.將準備要下達的直接指令收集成串，存入PLC檔中。
2.取出PLC檔中的直接指令依序的連續執行，直到檔案結束為止。
3.PLC檔控制動作相當於半自動的操作程序。

第三級的操控動作(CAI檔控制)：

1.將操作程序和各項畫面以CAI檔的格式作儲存。
2.取出CAI檔中的各項設定，以圖控畫面作顯示，並接受滑鼠和鍵盤的控制。
3.CAI檔控制動作相當於全自動的操作程序。

第四級的操控動作(CALLC程式控制)：

1.將要執行的程序，以C程式撰寫，並建立DOS環境中的EXE檔。
2.在fPLC中以CALLC指令呼叫來執行該項EXE檔。CALLC程式可利用公用記憶體和外部命令來取得fPLC中的各項數據並作操控。
3.CALLC程式控制可建立直接指令未提供的任何操作指令，提供更有彈性的操控動作。但是每次呼叫CALLC程式都只能執行一次，連續操作必須重複呼叫。

第五級的操控動作(CFUN程式控制)：

1.將要執行的程序，以C程式撰寫，並建立DOS環境中的DLL檔。
2.在fPLC中以CFUN指令呼叫來載入該DLL檔。CFUN程式可利用公用記憶體和外部命令來取得fPLC中的各項數據並作操控。
3.CFUN指令不只會在呼叫時執行，還會常駐在fPLC環境下隨著Timer呼叫作持續性的執行動作，可以建立長時間的操控程式，直到關閉該程式為止。
4.換句話說，CALLC程式和CFUN程式的基本差異是：
  在CALLC程式執行時，fPLC程式必須暫停，等CALLC結束後才能繼續執行。
  在CFUN程式執行時，fPLC程式還是繼續執行，兩者可以同時存在。

第六級的操控動作(C++程式控制)：

1.將要執行的程序，以C++程式撰寫，並建立WIN環境中的EXE檔。
2.C++程式的設計沒有限制，可任意建立操作畫面和操作程序。
3.當C++程式執行時，fPLC程式可隱藏在幕後，C++程式可利用公用記憶體和外部命令來取得fPLC中的各項數據並作操控。
4.換句話說，C++程式可以取得所有的操作控制權，可以建立fPLC程式未提供的任何操作畫面，提供最有彈性的操控動作。

設計者可以根據實際需求和撰寫時限，決定採用不同等級的操控設計方案。

6.1 CALLC 程式部分:

CALLC程式的撰寫程序

CALLC程式是在DOS環境下操作的EXE程式。撰寫程序如下：

1.進入[Microsoft Visual C++]環境。
2.執行[File][New][Project][Win32 Console Application]，並設定
    工作目錄：Location
    檔案名稱：Project Name
  其他未提及的選項都以預設值處理，請勿更動。
3.在[Step 1 of 1]中，選擇[An empty project]。
4.以任何文書編輯程式來撰寫C程式並儲存在工作目錄下。
5.執行[Project][Add To Project][Files]，再選擇
    已撰寫並放置在工作目錄下的C程式
  將指定程式加入project中。
6.選擇[FileView][Workspace][Source Files]，即可看到剛加入的C程式。連按兩下即可開始編輯C程式。
7.執行[Build][Set Active Configuration]，再選擇
    [Win32 Release]
  即可設定編譯環境。
8.執行[Build][Rebuild All]即可進行編譯。若無錯誤訊息即是編譯成功。
9.編譯結果會產生[EXE]型式的執行檔，可在DOS環境中直接執行。

在fPLC中執行CALLC程式是以直接指令型式執行。指令格式如下：

```
CALLC <filename> <p1> <p2> ...;        //**執行指定檔名的EXE程式
                                       //**相當於在DOS環境中下達
                                       //**<filename> <p1> <p2> ...的指令
```

CALLC程式在呼叫後只會執行一次，執行完畢立即結束。再度執行就必須再度呼叫CALLC指令。

第一個CALLC程式

檔案名稱：testCallC.c

```
/********************************************************
  filename: testCallC.c
                      designed by Pei-Chong Tang, Aug. 2005
********************************************************/
#include <windows.h>
#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <string.h>
#include <time.h>
//*******************************************************
// basic subprograms for common memory interface
//*******************************************************
static HANDLE hMap=NULL;
static LPSTR  TSR_data=NULL; //** pointer to PLC_data[]
static LPWORD PLC_data,CNC_data,DAS_data,MEM_data,RAM_data,ROM_data;
static LPWORD NET_data[31];
static LPSTR  PLC_command,PLC_echo;
                             //***** open common memory *****
static void open_common_memory(void)
{    int n;
     hMap=OpenFileMapping(FILE_MAP_WRITE,TRUE,
                  (LPSTR)"fPLC_common_memory");
     if (!hMap)     {TSR_data=NULL;             return;}
     TSR_data=MapViewOfFile(hMap,FILE_MAP_ALL_ACCESS,0,0,0);
     if (!TSR_data) {CloseHandle(hMap); hMap=0; return;}
     PLC_data = (LPWORD)TSR_data;
     CNC_data = PLC_data+(0x08000);
     DAS_data = PLC_data+(0x10000);
     MEM_data = PLC_data+(0x14000);
     RAM_data = PLC_data+(0x18000);
     ROM_data = PLC_data+(0x1c000);
     for (n=0; n<=30; n++) NET_data[n]=PLC_data+(0x20000+0x1000*n);
     PLC_command=(LPSTR)(PLC_data+0x3f000);
     PLC_echo   =(LPSTR)(PLC_data+0x3f800);
}
static void close_common_memory(void)
{    int n;
     if (hMap) {UnmapViewOfFile(TSR_data); CloseHandle(hMap);}
```

```
    hMap=0; TSR_data=NULL;
    PLC_data=CNC_data=DAS_data=MEM_data=RAM_data=ROM_data=NULL;
    for (n=0; n<32; n++) NET_data[n]=NULL;
    PLC_command=PLC_echo=NULL;
}
/*=========================================================
  DOS command format :
      testCallC <x> <y> <z>;
=========================================================*/
void main(int argc,char **argv)
{    static int x,y,z;
     x=0; y=1; z=2;            //** get parameters
     if (argc>=2) sscanf(argv[1],"%d",&x);
     if (argc>=3) sscanf(argv[2],"%d",&y);
     if (argc>=4) sscanf(argv[3],"%d",&z);
     open_common_memory();     //** do test function
     if (TSR_data)
     {    PLC_data[z]=PLC_data[x]+PLC_data[y];
     }
     close_common_memory();
}
```

其中：

1.建立兩項副程式，包括：
   open_common_memory() ：啓動指定的公用記憶體。
   close_common_memory()：關閉指定的公用記憶體。
2.一旦公用記憶體建立後，就可自由使用
   PLC_data[]   ：泛用的公用記憶體。
   CNC_data[]等 ：特定的公用記憶體區域。
   NET_data[][] ：指定節點的網路變數區。
   PLC_command[]：下達外部命令的區域。
   PLC_echo[]   ：記錄外部命令的執行結果。
3.如果公用記憶體宣告失敗，則會設定TSR_data=NULL，可用作檢知訊號。
4.在主程式中，首先取得x/y/z三項位址，再執行
   PLC_data[z] = PLC_data[x] + PLC_data[y]
   的工作。所以測試時只要在fPLC中檢測三項變數的執行結果即可。
5.在不同應用時，只要更改黃色部分即可。其他部分的程式都是固定格式，照抄即可。

CALLC程式的設計原則

CALLC程式是在DOS環境下執行的EXE檔案，所以任何DOS環境下的標準C程式的功能都可直接應用而不受限制。唯一需要注意的是CALLC程式最好在執行完畢後就立刻結束，儘量不要以迴圈型式作等待動作。這樣可以加速fPLC的程式執行效率。

6.2 CFUN 程式部分:

CFUN程式的撰寫程序

CFUNC程式的是在DOS環境下操作的DLL程式。撰寫程序如下：

1.進入[Microsoft Visual C++]環境。
2.執行[File][New][Project][Win32 Dynamic-Link Library]，並設定
　工作目錄：Location
　檔案名稱：Project Name
其他未提及的選項都以預設值處理，請勿更動。
3.在[Step 1 of 1]中，選擇[An empty DLL project]。
4.以任何文書編輯程式來撰寫C程式並儲存在工作目錄下。
5.執行[Project][Add To Project][Files]，再選擇
　已撰寫並放置在工作目錄下的C程式
將指定程式加入project中。
6.選擇[FileView][Workspace][Source Files]，即可看到剛加入的C程式。連按兩下即可開始編輯C程式。
7.執行[Build][Set Active Configuration]，再選擇
　[Win32 Release]
即可設定編譯環境。
8.執行[Build][Rebuild All]即可進行編譯。若無錯誤訊息即是編譯成功。
9.編譯結果會產生[DLL]型式的執行檔，必須經由fPLC呼叫才能執行。

在fPLC中執行CALLC程式是以直接指令型式執行。指令格式如下：

```
CFUN <name> <filename>;       //**啓動指定的CFUN程式
CFUN <name> CLOSE;            //**關閉指定的CFUN程式
CFUN CLEAR;                   //**清除所有的CFUN程式
CFUN;                         //**顯示所有的CFUN程式
CFUN <name> <p1> <p2> ...;    //**執行指定的CFUN程式
```

CFUN一旦啟動之後，就會持續的被重複執行，直到關閉CFUN後才停止。除了重複性的執行之外，CFUN也可被臨時呼叫而執行。

第一個CFUN程式

檔案名稱：testCFUN.c

```
/********************************************************
  a simple example for CFUN program (.DLL file)
                     designed by Pei-Chong Tang, March 2005
********************************************************/
#include <windows.h>
#include <stdio.h>
#include <string.h>
//*******************************************************
// basic subprograms for common memory interface
//*******************************************************
static void   CFUN_main(int,char **);
static BOOL   CFUN_timer(void);
static HANDLE hMap=NULL;
static LPSTR  TSR_data=NULL; //** pointer to PLC_data[]
static LPWORD PLC_data,CNC_data,DAS_data,MEM_data,RAM_data,ROM_data;
static LPWORD NET_data[31];
static LPSTR  PLC_command,PLC_echo,CFUN_data;
                              //***** open common memory *****
static void open_common_memory(void)
{    int n;
     hMap=OpenFileMapping(FILE_MAP_WRITE,TRUE,
                  (LPSTR)"fPLC_common_memory");
     if (!hMap)     {TSR_data=NULL;             return;}
     TSR_data=MapViewOfFile(hMap,FILE_MAP_ALL_ACCESS,0,0,0);
     if (!TSR_data) {CloseHandle(hMap); hMap=0; return;}
     PLC_data = (LPWORD)TSR_data;
     CNC_data = PLC_data+(0x08000);
     DAS_data = PLC_data+(0x10000);
     MEM_data = PLC_data+(0x14000);
     RAM_data = PLC_data+(0x18000);
     ROM_data = PLC_data+(0x1c000);
     for (n=0; n<=30; n++) NET_data[n]=PLC_data+(0x20000+0x1000*n);
     PLC_command =(LPSTR)(PLC_data+0x3f000);
     PLC_echo    =(LPSTR)(PLC_data+0x3f800);
     CFUN_data   =(LPSTR)(PLC_data+0x13400);
}

static void close_common_memory(void)
{    int n;
```

```
    if (hMap) {UnmapViewOfFile(TSR_data); CloseHandle(hMap);}
    hMap=0; TSR_data=NULL;
    PLC_data=CNC_data=DAS_data=MEM_data=RAM_data=ROM_data=NULL;
    for (n=0; n<32; n++) NET_data[n]=NULL;
    PLC_command=PLC_echo=CFUN_data=NULL;
}

//************************************************************
// entry points of standard DLL module
//************************************************************
                              //***** DLL interface *****
BOOL APIENTRY DllMain(HINSTANCE hInst, DWORD message, LPVOID lpReserved)
{   switch (message)
    {
    case DLL_PROCESS_ATTACH:          //**FileOpen()
         open_common_memory();
         CFUN_main(0,NULL);
         break;
    case DLL_PROCESS_DETACH:          //**FileClose()
         close_common_memory();
         break;
    case DLL_THREAD_ATTACH:
         break;
    case DLL_THREAD_DETACH:
         break;
    }
    return(TRUE);
}
                              //***** CFUN_timer() entry point *****
_declspec (dllexport) BOOL CFUN_time_program(void)
{   if (!TSR_data) return(TRUE);
    return(CFUN_timer());
}
                              //***** CFUN_main() entry point *****
_declspec (dllexport) void CFUN_main_program(void)
{   if (!TSR_data) return;
    CFUN_main((int)(*CFUN_data),(char **)(CFUN_data+4));
}

//************************************************************
// test function
//************************************************************
static int x=0,y=1,z=2;
                              //***** main program *****
```

```
static void CFUN_main(int argc,char **argv)
{     if (argc>=2) sscanf(argv[1],"%d",&x);
      if (argc>=3) sscanf(argv[2],"%d",&y);
      if (argc>=4) sscanf(argv[3],"%d",&z);
      PLC_data[z]=PLC_data[x]+PLC_data[y];
}
                                    //***** timer program *****
static BOOL CFUN_timer(void)
{     PLC_data[x]++;
      return(FALSE);
}
```

其中：

1.建立兩項副程式，包括：
   open_common_memory() ：啓動指定的公用記憶體。
   close_common_memory()：關閉指定的公用記憶體。
2.一旦公用記憶體建立後，就可自由使用
   PLC_data[]　　：泛用的公用記憶體。
   CNC_data[]等 ：特定的公用記憶體區域。
   NET_data[][] ：指定節點的網路變數區。
   PLC_command[]：下達外部命令的區域。
   PLC_echo[]　　：記錄外部命令的執行結果。
3.如果公用記憶體宣告失敗，則會設定TSR_data=NULL，可用作檢知訊號。
4.建立三項DLL檔案的進入點，包括：
   DllMain()：標準的DLL進入點，當啓動或關閉DLL檔時就會呼叫DllMain()。
   CFUN_time_program()：自訂的DLL進入點，由fPLC程式作重複性的呼叫。
   CFUN_main_program()：自訂的DLL進入點，由fPLC程式作指令性的呼叫。
5.對於應用程式而言，只需面對黃色部分的兩個副程式，包括：
   CFUN_main()：由直接指令下達的呼叫，參數和一般C程式中的main()一致。
   CFUN_timer()：重複性的自動呼叫，沒有輸入參數。在程式返回時，
      return(FALSE)：例行性的結束，沒有特別意義。
      return(TRUE) ：最後一次的呼叫，當返回fPLC後，fPLC會自動關閉這個DLL檔案。
6.在CFUN_main()中，首先取得x/y/z三項位址，再執行下述工作。
   PLC_data[z] = PLC_data[x] + PLC_data[y]
7.在CFUN_timer()中，將PLC_data[x]作重複性的遞增工作。
8.在不同應用時，只要更改黃色部分即可。其他部分的程式都是固定格式，照抄即可。

CFUN程式的設計原則

CFUN程式是在DOS環境下執行的DLL檔案,所以大部分DOS環境下的C程式功能都可直接應用。但是因為CFUN程式是由fPLC程式所呼叫的副程式,所以最好不要使用按鍵輸入和螢幕輸出。

CFUN程式和fPLC程式的參數連結方式有三種:

1. fPLC程式可藉由CFUN_main()的呼叫,而傳遞一些文字命令。
2. CFUN程式可透過公用記憶體而直接讀寫其中的所有資料。讀寫公用記憶體時是立即性的反應,不用等待。但是無法操控USB界面而讀寫DSP核心中的資料。
3. CFUN程式可透過外部命令而直接的操作fPLC,可讀寫DSP核心中的所有參數。只是因爲USB界面和光纖網路會產生延遲現象,讀取資料可能無法立即送達。

當CFUN程式在等待資料的送達、或是等待命令的執行結束時,如果用一般的迴圈方式作等待時,會造成PC端的操作凍結,嚴重地影響其他程式的操作。這時可利用CFUN_timer()的功能,作多工式重複掃瞄工作,這樣在等待期間就能讓其他視窗程式繼續作業,可大幅降低操作凍結的現象。

以CFUN執行的影像處理範例

這是一個變複雜的CFUN範例，負責將eM_IMAGE板上所記錄的影像資料，完整的傳入PC端並作顯示。其中的細部程序說明如下：

1. 首先下達命令給eM_IMAGE板，讓(640*480)的影像完整記錄在DSP核心的外部RAM中。在單色的需求下，需要640*480=307K-byte的空間。
2. 影像的掃瞄速度爲60Hz，再加上命令的同步問題，需要等待40ms左右的時間。
3. 在確認影像記錄完畢後，接著就要下達Move指令，將DSP核心中的RAM資料上載至PC端的公用記憶體中。每筆的Move指令可傳遞16K-word的資料，需500ms，必須等待上載資料確實送達之後，才能下達第二筆的Move指令，如此必須重複10次才能完整的取得所有資料。
4. 將上載後的影像資料以BMP檔格式作儲存，並利用PLOT指令在fPLC中繪出。
5. 整個程序結束後，即可自動關閉DLL程式。

當然這只是範例程式，如果對影像資料的處理方式有著不同的看法和做法，可以自行定義操作方式並撰寫CFUN程式，即可執行不同的動作和功能。

檔案名稱：CfunGetIMG.c

```
/*********************************************************
  CFUN program to get image file from eM_IMAGE board

                     designed by Pei-Chong Tang, Aug. 2005
*********************************************************/
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

//********************************************************
// basic subprograms for common memory interface
//********************************************************
static void   CFUN_main(int,char **);
static BOOL   CFUN_timer(void);
static HANDLE hMap=NULL;
static LPSTR  TSR_data=NULL; //** pointer to PLC_data[]
static LPWORD PLC_data,CNC_data,DAS_data,MEM_data,RAM_data,ROM_data;
static LPWORD NET_data[31];
static LPSTR  PLC_command,PLC_echo,CFUN_data;
                             //***** open common memory *****
static void open_common_memory(void)
{     int n;
```

```
    hMap=OpenFileMapping(FILE_MAP_WRITE,TRUE,
                (LPSTR)"fPLC_common_memory");
    if (!hMap)     {TSR_data=NULL;            return;}
    TSR_data=MapViewOfFile(hMap,FILE_MAP_ALL_ACCESS,0,0,0);
    if (!TSR_data) {CloseHandle(hMap); hMap=0; return;}
    PLC_data = (LPWORD)TSR_data;
    CNC_data = PLC_data+(0x08000);
    DAS_data = PLC_data+(0x10000);
    MEM_data = PLC_data+(0x14000);
    RAM_data = PLC_data+(0x18000);
    ROM_data = PLC_data+(0x1c000);
    for (n=0; n<=30; n++) NET_data[n]=PLC_data+(0x20000+0x1000*n);
    PLC_command =(LPSTR)(PLC_data+0x3f000);
    PLC_echo    =(LPSTR)(PLC_data+0x3f800);
    CFUN_data   =(LPSTR)(PLC_data+0x13400);
}
static void close_common_memory(void)
{   int n;
    if (hMap) {UnmapViewOfFile(TSR_data); CloseHandle(hMap);}
    hMap=0; TSR_data=NULL;
    PLC_data=CNC_data=DAS_data=MEM_data=RAM_data=ROM_data=NULL;
    for (n=0; n<32; n++) NET_data[n]=NULL;
    PLC_command=PLC_echo=CFUN_data=NULL;
}
//*********************************************************
// entry points of standard DLL module
//*********************************************************
                              //***** DLL interface *****
BOOL APIENTRY DllMain(HINSTANCE hInst, DWORD message, LPVOID lpReserved)
{   switch (message)
    {
    case DLL_PROCESS_ATTACH:          //**FileOpen()
        open_common_memory();
        CFUN_main(0,NULL);
        break;
    case DLL_PROCESS_DETACH:          //**FileClose()
        close_common_memory();
        break;
    case DLL_THREAD_ATTACH:
        break;
    case DLL_THREAD_DETACH:
        break;
    }
    return(TRUE);
```

```
}                              //***** CFUN_timer() entry point *****
_declspec (dllexport) BOOL CFUN_time_program(void)
{    if (!TSR_data) return(TRUE);
     return(CFUN_timer());
}                              //***** CFUN_main() entry point *****
_declspec (dllexport) void CFUN_main_program(void)
{    if (!TSR_data) return;
     CFUN_main((int)(*CFUN_data),(char **)(CFUN_data+4));
}

//************************************************************
// functions about PLC_command[] setting
//************************************************************
static char *sPLC;
                               //***** command finished ?? *****
static BOOL PLC_finished(void)
{    if (*PLC_command) return(FALSE);
     return(TRUE);
}                              //***** reset to null command *****
static void PLC_reset(void)
{    sPLC=PLC_command; *sPLC=0;
}                              //***** append next command *****
static void PLC_append(char *s)
{    while (*s) *sPLC++=*s++;
     *sPLC=0;
}                              //***** put SET command *****
static void PLC_Scommand(int addr,int dat)
{    char buf[81]; sprintf(buf,"S %d %d;",addr,dat);
     PLC_append(buf);
}                              //***** put MOVE command *****
static void PLC_Mcommand(char *dir,char *area,int addr,int len)
{    char buf[81]; sprintf(buf,"M %s %d %d %s;",dir,addr,len,area);
     PLC_append(buf);
}                              //***** put MOVE page code *****
static void PLC_PageCode(int code)
{    char buf[81]; sprintf(buf,"M PAGE 0x%04X;",code);
     PLC_append(buf);
}
//============================================================
// functions about bitmap file
//============================================================
static WORD    IMG_data[640*480];
static RGBQUAD IMG_bits[640*480];
                    //***** transfer to RGB format *****
```

```
static void get_bitmap(void)
{    RGBQUAD *p; int k,x,y,xsize,ysize; WORD d;
     xsize=640; ysize=480;
     for (y=0,k=0; y<ysize; y++)
     {    p=IMG_bits+xsize*(ysize-y-1);
          for (x=0; x<xsize; x+=2,k++,p++)
          {    d=IMG_data[k]&0xff;
               p->rgbBlue =(char)d;
               p->rgbGreen=(char)d;
               p->rgbRed  =(char)d;
               p->rgbReserved=0;
               d=(IMG_data[k]>>8)&0xff; p++;
               p->rgbBlue =(char)d;
               p->rgbGreen=(char)d;
               p->rgbRed  =(char)d;
               p->rgbReserved=0;
          }
     }
}
                   //***** save to bitmap file *****
static void save_bitmap(void)
{    static BITMAPFILEHEADER bmfh; static BITMAPINFO bmi;
     int fh,xsize,ysize; static OFSTRUCT of;
                             //**open file
     xsize=640; ysize=480;
     if ((fh=OpenFile((LPSTR)"testBMP.bmp",&of,OF_CREATE))<=0) return;
     bmfh.bfType=19778;      //**write BITMAPFILEHEADER, BM=19778
     bmfh.bfSize=sizeof(BITMAPFILEHEADER)
               +sizeof(BITMAPINFO)+xsize*ysize*4;
     bmfh.bfReserved1=0;
     bmfh.bfReserved2=0;
     bmfh.bfOffBits=sizeof(BITMAPFILEHEADER)+sizeof(BITMAPINFO);
     _lwrite(fh,(LPSTR)&bmfh,sizeof(BITMAPFILEHEADER));
                             //**write BITMAPINFO
     bmi.bmiHeader.biSize    =sizeof(BITMAPINFOHEADER);
     bmi.bmiHeader.biWidth   =xsize;
     bmi.bmiHeader.biHeight  =ysize;
     bmi.bmiHeader.biPlanes  =1;
     bmi.bmiHeader.biBitCount=32;
     bmi.bmiHeader.biCompression  =BI_RGB;
     bmi.bmiHeader.biSizeImage    =0;
     bmi.bmiHeader.biXPelsPerMeter=5000;
     bmi.bmiHeader.biYPelsPerMeter=5000;
     bmi.bmiHeader.biClrUsed      =0;
```

```
    bmi.bmiHeader.biClrImportant =0;
    bmi.bmiColors[0].rgbBlue=0;
    bmi.bmiColors[0].rgbGreen=0;
    bmi.bmiColors[0].rgbRed=0;
    bmi.bmiColors[0].rgbReserved=0;
    _lwrite(fh,(LPSTR)&bmi, sizeof(BITMAPINFOHEADER));
                            //**write IMAGE
    get_bitmap();
    _lwrite(fh,(LPSTR)IMG_bits,xsize*ysize*4);
    _lclose(fh);
}
//*******************************************************
// CFUN control process
//*******************************************************
#define aIMG_vcmd                   0x84
#define RAM_length ((short)(PLC_data[32741]))
static int status=0;
                            //***** main program *****
static void CFUN_main(int argc,char **argv)
{   status=1;
}
                            //***** timer program *****
static BOOL CFUN_timer(void)
{   static DWORD t0=0; static int i,k,len,dt,dly,page,npage;
    if (!PLC_finished()) return(FALSE); //**wait command finished
    PLC_reset();                        //**reset to null command
    dt=GetCurrentTime()-t0;             //**get time difference
    switch (status)
    {
    case 1 :
        PLC_Scommand(aIMG_vcmd,1);          //**enable image
        t0=GetCurrentTime(); dly=100;       //**delay 100ms
        len=640*480/2; npage=(len+16383)/16384;
        k=0; page=0;                        //**from page #0
        status++;   break;
    case 2 :
        if (dt<dly) break;                  //**wait a while
        PLC_Scommand(aIMG_vcmd,0);          //**disable image
        PLC_PageCode(0xf800+(page<<4));     //**set page code
        PLC_Mcommand("UP","RAM",0,16384);   //**move 16K-word
        t0=GetCurrentTime(); dly=100;       //**delay 100ms
        status++; break;
    case 3 :
        if (dt<dly || RAM_length>0) break;  //**wait a while
```

```
        for (i=0; i<16384 && k<len; i++,k++) //**store image data
            IMG_data[k]=RAM_data[i];
        if ((++page)>=npage) {status++; break;}
        PLC_PageCode(0xf800+(page<<4));      //**set page code
        PLC_Mcommand("UP","RAM",0,16384);    //**move 16K-word
        t0=GetCurrentTime(); dly=100;        //**delay 100ms
        break;
    case 4 :
        save_bitmap();                       //**save to file
        PLC_append("PLOT scope 0;");         //**plot bitmap
        PLC_append("PLOT redraw;");
        PLC_append("PLOT bitmap 10 10 160 140 testBMP.bmp;");
        status=9; break;
    case 9 :
        status=0; return(TRUE);
    default:
        status=0; break;
    }
    return(FALSE);
}
```

其中：

1.程式可概分成四段作說明，包括：
   標準的CFUN格式，
   外部指令處理用的副程式，
   影像檔(BMP檔)處理用的副程式，和
   整個程序控制的主程式。

2.標準的CFUN格式包括了下述五個程式，請勿隨意更改。

```
open_common_memory() ：啓動指定的公用記憶體。
close_common_memory()：關閉指定的公用記憶體。
DllMain()            ：DLL程式的進入點。
CFUN_time_program()  ：重複性呼叫的進入點。
CFUN_main_program()  ：指令性呼叫的進入點。
```

3.外部指令是藉由PLC_command[]區的設定而控制fPLC程式，可將要下達的直接指令全部串接起來，整批送給fPLC程式作處理。爲了方便程式的撰寫，在這個範例中整理了一些好用的副程式，包括：

```
BOOL PLC_finished()：檢查上批的外部指令是否已經完全結束。
void PLC_reset()   ：清除既有的PLC_command[]區。
void PLC_append()  ：增加一筆指令至PLC_command[]區中。
void PLC_Scommand()：增加一筆 Set指令至PLC_command[]區中。
void PLC_Mcommand()：增加一筆Move指令至PLC_command[]區中。
void PLC_PageCode()：增加一筆M_PAGE指令至PLC_command[]區中。
```

4.在應用時的執行程序如下：

A.以PLC_finished()來檢查上批的外部指令是否結束，結束後才准繼續執行。
B.執行PLC_reset()來清除PLC_command[]區。
C.依需要將新指令逐筆加入PLC_command[]區中。
D.不需再作任何處理，即可結束程式。當fPLC程式由PLC_command[]區中取得外部指令後，會立即處理所有指令，執行完畢後會將PLC_command[]清空。
E.外部指令的執行結果會儲存在PLC_echo[]中，但通常不需處理PLC_echo[]，而是直接讀取公用變數區來檢驗處理結果。

5.影像檔處理用的副程式包括：

```
void get_bitmap()：將上載後的影像資料(已儲存在IMG_data[]中)轉換成
                   RGB格式並儲存在IMG_data[]中。
void save_bitmap()：將影像資料以BMP檔格式儲存在testBMP.bmp中。
```

6.而真正作程序處理的程式包括：

```
void CFUN_main() ：指令性的程式呼叫處理。
BOOL CFUN_timer()：重複性的程式呼叫處理。
```

其中基本的執行架構如下：

A.建立一個標準的執行旗標(status)，指示目前的程序執行狀態。
B.程式啓動時設定(status=1)，再根據status值而依序執行不同的程序處理。
C.在程式結束後設定(status=0)。

7.所以在範例中：

A.當CFUN_main()執行時，設定(status=1)。
B.而在CFUN_timer()執行時，根據status值而執行不同程序的處理。
C.在執行時如果需要等待一段時間，可以利用GetCurrentTime()來取得目前的絕對時間(以ms爲單位)，以決定要繼續等待或是等待結束。
D.在執行時如果需要下達外部指令，可以利用已提供的幾組副程式。

8.範例中的執行程序如下：

status=1：程序開始，處理程序包括：
執行[S IMG_vcmd 1;]指令來啓動eM_IMAGE板。
設定t0=目前時間，而dly=100，即設定等待時間爲100ms。
設定len=640*480/2;即影像容量爲153600-word。
設定npage爲M指令的總次數，153K共需10組M指令才能完成。
設定k=0; page=0;其中k爲word總數而page爲M指令總數。
執行完畢後直接進入status=2的狀態。

status=2：等待100ms的結束。在等待結束後執行

執行[S IMG_vcmd 0;]指令來停止eM_IMAGE板並取得RAM的控制權
執行[M PAGE code;]指令來設定RAM的頁數。
執行[M UP 0 16384 RAM;]指令來上載RAM資料，一次16K-word。
設定t0=目前時間，而dly=100，即設定等待時間爲100ms。
設定完畢後直接進入status=3的狀態。

status=3：等待100ms並檢查RAM_length。RAM_length記錄Move指令中未完成的上載word數，當RAM_length=0時即確認所有上載資料都已傳達。在等待結束且上載資料都已到達後，才准繼續執行

將上載資料從RAM區轉移至RAM_dat[]中。
更新k值和page值，當page>=npage時進入status=4的狀態。
否則繼續執行下述程序，包括：
執行[M PAGE code;]指令來設定RAM的頁數。
執行[M UP 0 16384 RAM;]指令來上載RAM資料，一次16K-word。
設定t0=目前時間，而dly=100，即設定等待時間爲100ms。
設定完畢後繼續維持status=3的狀態。

status=4：上載結束，可依序執行下述程序：

執行save_bitmap()將上載影像儲存在testBMP.bmp中。
執行[PLOT scope 0;]將示波器視窗設爲繪圖模式。
執行[PLOT redraw;] 來清除示波器視窗。
執行[PLOT bitmap x0 y0 dx dy filename;]來顯示BMP檔。
設定完畢後直接進入status=9的狀態。

status=9：CFUN結束，程序包括：

設定status=0來標示程序結束。
以return(TRUE)來結束CFUN程式。

9.一般而言，CFUN_timer()是個重複性的呼叫指令，在fPLC中約以10ms的週期來執行，但是週期並不穩定。由於CFUN_timer()並不是多工性的重複處理，而是fPLC程式中的主動呼叫，所以可以完全確認

A.當CFUN_timer()程式運行時，fPLC程式不會同步工作。
B.一定要等CFUN_timer()呼叫結束後，才會返回fPLC程式中繼續執行。
C.在CFUN_timer()程式結束時，
return(FALSE)：是正常結束方式，fPLC將繼續執行下一次的呼叫。
Return(TRUE) ：通知fPLC結束這組CFUN程式，以後將不再呼叫。

在fPLC中只要執行：

```
CFUN IMG CfunGetIMG.dll;        //**啓動指定的CFUN程式
```

即可啟動這個影像處理程式，整個處理時間約為10秒，執行結束後會將影像存檔為testBMP.bmp，並直接在示波器視窗中顯示。

6.3 C++ 程式部分:

C++程式的撰寫程序

C++程式的是在Windows環境下操作的EXE程式。撰寫程序如下：

1.進入[Microsoft Visual C++]環境。
2.執行[File][New][Project][MFC AppWizard[exe]]，並設定
　工作目錄：Location
　檔案名稱：Project Name
其他未提及的選項都以預設值處理，請勿更動。
3.在[Step 1]中，選擇[Dialog based]。
4.按[Finish]即可開始編輯程式畫面。
5.執行[Build][Set Active Configuration]，再選擇
　[Win32 Release]
即可設定編譯環境。
6.在VC++環境中進行程式編輯。
7.執行[Build][Rebuild All]即可進行編譯。若無錯誤訊息即是編譯成功。
8.執行[Build][Execute ***.exe]即可執行C++程式。但必須特別提醒：必須先執行fPLC程式再執行C++程式，C++程式才能正確運用fPLC中的公用記憶體。

C++程式和一般C程式有著非常明顯的差異，說明如下：

1.一般C程式可利用任何文書編輯(例如筆記本或是TextPad)去建立和編輯C程式，而VC++環境只是執行單純的Compiler動作而已。
2.但是C++程式卻必須在VC++環境下，以指定的各項圖控工具來編輯程式。若以其他工具自行更改CPP程式，常會產生不可預料的嚴重後果，千萬不要輕易嘗試。
3.C++程式的設計以操作畫面為主體，分成下述三種觀點來規劃：
　Class　　：C++程式的控制主體
　Resource：畫面控制的整合管理
　File　　：各項程式的程式碼
但實際上三者所敘述的只是同一個程式的不同面貌而已。
4.基本上C++程式是以Class為主，說明如下：
A.操作畫面中的每個物件都是一個Class，有著獨立的Class名稱。
B.每個Class都必須面對視窗環境傳來的各項Message，代表目前視窗環境中的每個細部操控動作。因此在每個Class中都必須準備各項Member Function來處理不同的Message。
C.而一般的C程式只是對映到這些Member Function而已。

第一個C++程式

第一個C++的撰寫程序如下：

1.進入[Microsoft Visual C++]環境，新增一項Project，名稱爲testCPP。
2.以標準的C++新增程序，不需更改或撰寫任何程式就可得到預設的範例程式，可直接作編譯及執行。
3.這個範例程式爲一個Dialog視窗，內含兩個按鍵，分別爲"確定"和"取消"。
4.進入Dialog的編輯畫面，對兩個按鍵作點選，即可增加兩個按鍵所對映的Class。
5.在工作環境的視窗中，依序選擇
   [Files][testCPP files][Source Files][testCPP.cpp]
  即可看到testCPP.cpp的預設程式。程式看起來非常複雜，但不用特別理會。
6.執行[View][Class Wizard..]，即可看到各個Class的分佈和內涵。包括：
   CTestCPPDlg：整個Dialog所對映的Class
   IDCANCEL　：按鍵"取消"所對映的Class
   IDOK　　　：按鍵"確定"所對映的Class
  而每個Class都有其面對的Message和Member Function。
7.第一個範例程式所需增加的功能如下：
  A.在TestCPPDlg中，必須處理下述Message，包括
     ON_WM_INITDIALOG：對映到OnInitDialog()程式，代表程式啓動。
     ON_WM_CLOSE　　：對映到OnClose()程式，代表程式結束。
     ON_WM_TIMER　　：對映到OnTimer()程式，代表Timer的週期性執行。
  B.在IDOK中，必須處理下述Message
     ON_IDOK:BN_CLICKED：對映到OnOK()程式，代表"確定"按鍵的按下。
8.而所執行的工作包括：
  OnInitDialog()：啓動公用記憶體及Timer。
  OnClose()　　：結束公用記憶體及Timer。
  OnTimer()　　：執行(z=x+y)的計算。
  OnOK()　　　：執行(x=x+1)的計算。
9.下面將分別敘述各項工作的撰寫程序。

步驟#1：增加必要的Member Functions

1.執行[View][Class Wizard..]，即可看到各個Class的分佈和內涵。
2.如果Member Funcyions中已有所需要的項目，則不需處理。否則請點選指定的Message項目，並執行[Add Function]選項，即可增加該function。
3.依序進行，直到所需的function都已出現。應該包括：
OnInitDialog()
OnClose()
OnTimer()
OnOK()

步驟#2：增對每個function，撰寫所需要的程式。

1.在Member Functions中點選指定function，連按兩下即可出現CPP程式。
2.針對OnInitDialog()作下述的更改，其中黃色部分是新增的程式部分。

```
BOOL CFPLC_cprog_1Dlg::OnInitDialog()
{   CDialog::OnInitDialog();
    // Set the icon for this dialog.  ...
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);          // Set big icon
    SetIcon(m_hIcon, FALSE);         // Set small icon
    // TODO: Add extra initialization here
     open_common_memory();            //啓動公用記憶體
     m_x=10; m_y=11; m_z=12;          //設定參數位址
     SetTimer(1,10,NULL);             //啓動Timer程序
    return TRUE;  // return TRUE  unless you ...
}
```

3.針對OnClose()作下述的更改，其中黃色部分是新增的程式部分。

```
void CTestCPPDlg::OnClose()
{   // TODO: Add your message handler code here ...
     close_common_memory();
    KillTimer(1);
     CDialog::OnClose();
}
```

4.針對OnTimer()作下述的更改，其中黃色部分是新增的程式部分。

```
void CTestCPPDlg::OnTimer(UINT nIDEvent)
{   // TODO: Add your message handler code here ...
     if (TSR_data) {PLC_data[m_x]++;}
    CDialog::OnTimer(nIDEvent);
}
```

5.針對OnOK()作下述的更改，其中黃色部分是新增部分，而綠色是刪除的部分。

```
void CTestCPPDlg::OnOK()
{   // TODO: Add extra validation here
     if (TSR_data) {PLC_data[m_z]=PLC_data[m_x]+PLC_data[m_y];}
    //  CDialog::OnOK();
}
```

步驟#3：增加公用記憶體所需的處理程式。

1. 將下述處理公用記憶體的基本程式加在OnInitDialog()程式之前的空白區。

```
//==========================================================
//  functions about common memory
//==========================================================
static HANDLE hMap=0;                      //** common memory
static LPSTR  TSR_data=NULL;
static LPWORD PLC_data,CNC_data,DAS_data,MEM_data,RAM_data,ROM_data;
static LPWORD NET_data[31];
static LPSTR  PLC_command,PLC_echo;
static void open_common_memory(void)   //***** open common memory *****
{    int n;
     hMap=OpenFileMapping(FILE_MAP_WRITE,TRUE,
                         (LPSTR)"fPLC_common_memory");
     if (!hMap)     {TSR_data=NULL;             return;}
     TSR_data=(char *)MapViewOfFile(hMap,FILE_MAP_ALL_ACCESS,0,0,0);
     if (!TSR_data) {CloseHandle(hMap); hMap=0; return;}
     PLC_data = (LPWORD)TSR_data;
     CNC_data = PLC_data+0x08000;
     DAS_data = PLC_data+0x10000;
     MEM_data = PLC_data+0x14000;
     RAM_data = PLC_data+0x18000;
```

```
    ROM_data = PLC_data+0x1c000;
    for (n=0; n<=30; n++) NET_data[n]=PLC_data+(0x20000+0x1000*n);
    PLC_command=(LPSTR)&PLC_data[0x3f000];
    PLC_echo   =(LPSTR)&PLC_data[0x3f800];
}
static void close_common_memory(void)
{   if (hMap) {UnmapViewOfFile(TSR_data); CloseHandle(hMap);}
    hMap=0; TSR_data=NULL;
}
```

2. 將下述的參數定義加在OnInitDialog()程式之前的空白區。

```
static int m_x,m_y,m_z;
```

步驟#4：編譯及執行。

1.執行[Build][Rebuild All]即可進行編譯。若無錯誤訊息即是編譯成功。
2.先執行fPLC程式。
3.執行[Build][Execute ***.exe]即可執行C++程式。
4.由於在C++中設定：
  A.x=10, y=11, z=12
  B.每次Timer都執行
      PLC_data[x]++;
    所以PLC_data[10]會出現遞增值。
  C.每次按下確定都會執行
      PLC_data[z]=PLC_data[x]+PLC_data[y];
    所以每次按鍵後PLC_data[12]就會變更。
5.C++程式中並沒顯示執行結果，所以執行結果要由fPLC來觀測。程序如下：
  A.執行
```
RUN DSP #;      //選擇PC端的虛擬DSP
D 10 8;         //顯示R10~R17
```
  B.由於R10即PLC_data[10]，R12即PLC_data[12]，所以觀測其變化即可得知C++程式的執行結果。

C++程式的設計原則

C++程式是在WIN環境下執行的EXE檔案，所以所有WIN環境下的C++程式功能都可直接應用。由於C++程式可直接控制整個螢幕，所以所有的人機操作界面都可交由C++程式來執行，此時fPLC將轉成背景程式來使用。

C++程式和fPLC程式的參數連結方式有三種：

1.所有參數都由C++程式直接處理，不需fPLC程式來傳遞參數。
2.C++程式可透過公用記憶體而直接讀寫其中的所有資料。讀寫公用記憶體時是立即性的反應，不用等待。但是無法操控USB界面而讀寫DSP核心中的資料。
3.C++程式可透過外部命令而直接的操作fPLC，可讀寫DSP核心中的所有參數。只是因爲USB界面和光纖網路會產生延遲現象，讀取資料可能無法立即送達。

當C++程式在等待資料的送達、或是等待命令執行結束時，如果用一般的迴圈方式作等待時，會造成PC端的操作凍結，嚴重地影響其他程式的操作。所以在我們建議使用上述範例中的Timer的功能，作多工式重複掃瞄工作，在等待期間就能讓其他視窗程式繼續作業，可大幅降低操作凍結的現象。

C++程式可以依照操作需求作自由的撰寫，和fPLC之間的關係只有下述幾項：

1.在C++程式啓動時，啓動公用記憶體及Timer。
2.在C++程式結束時，關閉公用記憶體及Timer。
3.在Timer程序中可讀寫公用記憶體，並可透過外部指令來遙控fPLC的操作。
4.在按鍵下達時可對fPLC進行外部命令的遙控動作。
5.可遙控fPLC來進行DSP核心的讀取動作，並隨時更新操作畫面中的資料顯示。

附錄一.piDSP 系統規劃

piDSP界面規劃

唐佩忠

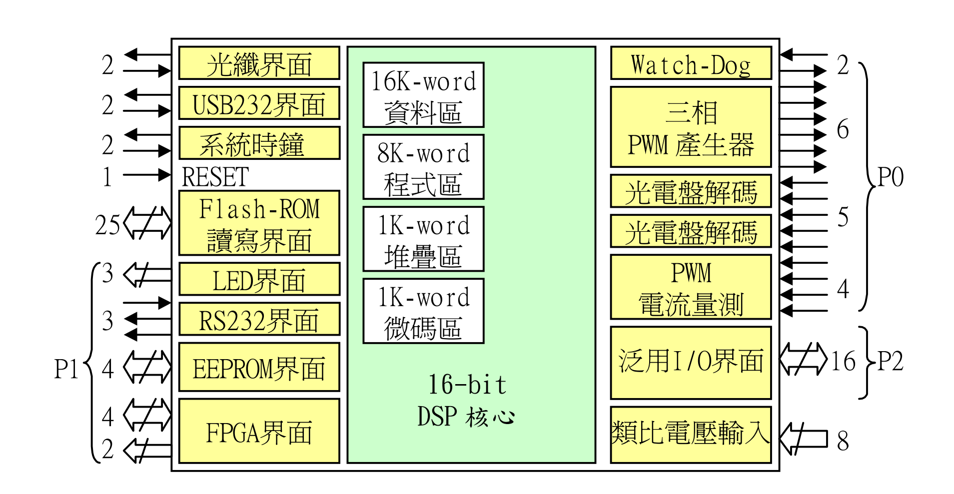
piDSP 是電源隔離式(Power-Isolated)的 DSP 元件，應用在低價位的變速馬達控制和高精度的伺服驅動市場。其特色包括：

1. 16-bit的 DSP 核心，運作於 24.576 MHz的工作頻率。
2. 直接連接 IM13100 等高功率 IGBT 模組，或是 IR213x 等閘極驅動 IC。提供三相/六組的 PWM 驅動訊號。
3. 直接連接 IR2172S 等數位式的電流偵測 IC。
4. 直接連接光電寫碼器，以及多工或非多工式的 U/V/W 霍爾相角回授訊號。
5. 提供非同步串列通訊界面(RS232X/RS422/RS485)。
6. 直接連接 APTX179A 和 APRX179A 等光發射及接收元件，提供光纖網路連線。
7. 光纖網路提供電源隔離式的發展和通訊環境，功能包括：
   - A. DSP 和功率驅動端採用同一電源系統，兩者之間不需要高速光耦合電路即可直接連線。
   - B. DSP 和開發工具之間採用光纖網路連線，可透過光纖網路來執行界面測試、程式下載、FlashROM/EEPROM 燒錄、線上監控和動態記錄及偵錯工作。
   - C. DSP 和其他 DSP 間採用光纖網路連線，可執行各軸伺服命令和回授傳遞的同步通訊工作。
   - D. 透過 USB 界面可和工業級 PC 連線，可執行多軸控制的產業控制應用。
   - E. 光纖通訊工作由 DSP 中的硬體電路執行，不會干擾 DSP 中即時控制程式的執行效率，也不需要程式設計者費心。
8. 直接連接 FlashROM 和 EEPROM，提供開機下載及參數設定的功能。
9. 直接連接 FPGA 元件作界面擴張用，提供 FPGA 電路下載及界面連線的功能。

系統規劃

界面規劃

piDSP界面的系統方塊圖如下：



其中：

1.DSP電源爲3.3V單電源，內含16-bit的DSP核心和32K-word位址範圍的RAM，包括資料區、程式區、堆疊區和暫存器。

2.伺服驅動界面包括：

- 三相PWM產生器：可直接驅動三相IGBT模組的閘極端。
- Watch-Dog　　：PWM輸出保護處理。
- PWM電流量測　：數位型式的電流量測，共四組。
- 光電盤解碼　　：光電盤的A/B/Z輸入，可接兩組光電盤。
- FPGA界面　　：連接擴張用FPGA的下載界面及通訊界面。
- 泛用I/O界面　：共16-bit，但其他I/O也可設爲泛用I/O，最多達48-bit。
- 類比電壓輸入　：10-bit精度的ADC輸入，共8組。

3.通訊界面包括：

- RS232界面　　：泛用型的RS232/RS422/RS485界面。
- 光纖通訊界面　：專用型的光纖網路界面，包括同步通訊和非同步通訊。
- USB232界面　：專用型的USB-232界面，用以連接個人電腦。

4.系統界面包括：

- FlashROM界面　：可連接FlashROM或SRAM，作爲程式儲存和FPGA下載用。
- EEPROM界面　：可連接EEPROM作爲重要參數的儲存。
- 系統時鐘　　：連接24.576MHz的震盪器。

5.界面共計89-pin，以100-pin的規格而言，還有11-pin留給電源和其他用途。

記憶體配置

記憶體共佔用32K-word的位址空間，包括了界面區、網路區、資料區、程式區和暫存器等等。位址分配如下：

位 址	長 度	名 稱	說 明
0x0010~0x001F	16-word	I/O界面區	對映到觸發的I/O界面
0x0020~0x007F	96-word	I/O界面區	對映到掃瞄的I/O界面
0x0100~0x017F	128-word	網路輸入區	對映到網路輸入記憶體(RxRAM)
0x0180~0x01FF	128-word	網路輸出區	對映到網路輸出記憶體(TxRAM)
0x0000~0x3FFF	16K-word	資料區	包括界面、變數、表格及堆疊
0x4000~0x5FFF	8K-word	程式區	儲存DSP程式
0x6000~0x6FFF	4K-word	記錄區	儲存測試記錄(模擬器專用)
0x7000~0x77FF	2K-word	微碼區	儲存微控碼
0x7800~0x7FFF	2K-word	暫存器	對映到DSP控制命令和狀態

觸發型的I/O界面共佔用16-word，位址分配如下：

0x10~0x17	OP0	IP0	OP1	IP1	OP2	IP2	?	?
0x18~0x1F	eeCMD	eeSTS	mmCMD	mmSTS	ppCMD	ppSTS	ssCMD	ssSTS

掃瞄型的I/O界面共佔用96-word，位址分配如下：

位址	CHN=0	CHN=1	CHN=2	CHN=3	CHN=4	CHN=5	CHN=6	CHN=7
0x20~0x27	IPB0	IPB1	IPB2	IPB3	IPB4	IPB5	IPB6	IPB7
0x28~0x2F	AMP0	AMP1	AMP2	AMP3	PHT0	PHT1	PHT1z	?
0x30~0x37	OPA0	OPA1	OPA2	OPA3	OPA4	OPA5	OPA6	OPA7
0x38~0x3F	PWM0	PWM1	PWM2	PWMc	PHTc	AMPc	?	?
0x40~0x47	IPC0	IPC1	IPC2	IPC3	IPC4	IPC5	IPC6	IPC7
0x48~0x4F	ADC0	ADC1	ADC2	ADC3	ADC4	ADC5	ADC6	ADC7
0x50~0x57	OPB0	OPB1	OPB2	OPB3	OPB4	OPB5	OPB6	OPB7
0x58~0x5F	IOMD0	IOMD1	?	?	?	?	?	netCMD
0x60~0x67	IPA0	IPA1	IPA2	IPA3	IPA4	IPA5	IPA6	IPA7
0x68~0x6F	?	?	?	?	?	?	?	netSTS
0x70~0x77	OPC0	OPC1	OPC2	OPC3	OPC4	OPC5	OPC6	OPC7
0x78~0x7F	MSK0	MSK1	MSK2	MSK3	MSK4	MSK5	MSK6	MSK7

其中：

1. 陰影部分對映到FPGA擴張界面，共計24-word的輸出阜和24-word的輸入阜。
2. AMPn/CNTn對映到四組的PWM電流偵測。ADCn對映到八組的ADC輸入，PHTn對映到兩組的光電盤輸入，而PWMn對映到六組的PWM輸出。
3. IPn/OPn對映到泛用型的I/O界面，而IOMn爲其中的模式選擇。
4. 目前此表格尚未完全定案??

暫存器共佔用4-word，定義如下：

位 址	模式	名 稱	說　明
0x7FF0	讀寫	dspTMR	DSP中斷的設定頻率：0=手動，1=1KHz，2=2KHz 3=4KHz，4=8KHz，5=16KHz，6=32KHz
0x7FF1	寫入	dspCMD	控制DSP的啓動與停止 b3=INH：當INH=1時強迫DSP立刻停止 b2=ENB：當ENB=1時啓動，當ENB=0時停止 b1=INT：當INT=1時產生手動中斷
0x7FF1	讀取	dspSTS	DSP目前的控制狀態 b5=TMR：計時中斷執行中 b4=LOP：主迴圈執行中 b3=INH，b2=ENB，b1=INT
0x7FF2	寫入	dasCMD	
0x7FF2	讀取	dasSTS	

I/O接腳共計87根，表列如下：

泛用I/O	長度	專用I/O	說　明
P0L0~5	6-bit	PWM0~5	三相PWM的控制輸出訊號
P0L6	1-bit	RST/	Watch-Dog的RST訊號輸入
P0L7	1-bit	ENB/	Watch-Dog的控制結果輸出
P0H0~3	4-bit	AMP0~3	電流回授的PWM訊號輸入
P0H4~5	2-bit	PHT0~1	光電寫碼器的A/B相輸入(第一組)
P0H6~7	2-bit	PHT2~3	光電寫碼器的A/B相輸入(第二組)
P2L0~7	8-bit	--	泛用I/O界面
P2H0~7	8-bit	--	泛用I/O界面
P1L0~3	4-bit	SIN/SOUT SCLK/CS	界面擴張用FPGA的通訊界面
P1L4~7	4-bit	PROG/DONE DIN/CCLK	界面擴張用FPGA的下載界面
P1H0~3	4-bit	SI/SO/ SCK/CS	EEPROM界面讀寫訊號
P1H4~7	4-bit	A17~14	FlashROM位址中的4-bit高位元
--	14bit	A0~A13	FlashROM中的位址匯流排
--	8-bit	D0~D7	FlashROM中的資料匯流排
--	2-bit	WR/RD	FlashROM中的讀寫控制
--	2-bit	TxD/ RxD	RS232界面訊號
--	2-bit	TxP/RxP	光纖網路界面訊號
--	2-bit	OCSi/OSCo	系統時鐘界面(24.576MHz)
--	1-bit	RESET/	系統復歸訊號
--	8-bit	ADC0~7	八頻道的類比輸入訊號

模擬	接頭	泛用	專 用 定 義	模擬	接頭	泛用	專 用 定 義
P3A0	J2-3	P0.0	PWM0H(O)	P3C0	J2-19	--	FPGA擴張點
P3A1	J2-4	P0.1	PWM1H(O)	P3C1	J2-20	--	FPGA擴張點
P3A2	J2-5	P0.2	PWM2H(O)	P3C2	J2-21	--	FPGA擴張點
P3A3	J2-6	P0.3	PWM0L(O)	P3C3	J2-22	--	FPGA擴張點
P3A4	J2-7	P0.4	PWM1L(O)	P3C4	J2-23	--	FPGA擴張點
P3A5	J2-8	P0.5	PWM2L(O)	P3C5	J2-24	--	FPGA擴張點
P3A6	J2-9	P0.6	PWMENB(O)	P3C6	J2-25	--	FPGA擴張點
P3A7	J2-10	P0.7	PHT1Z(I)	P3C7	J2-26	--	FPGA擴張點
P3B0	J2-11	P0.8	PHT1A(I)				
P3B1	J2-12	P0.9	PHT1B(I)				
P3B2	J2-13	P0.10	PHT0A(I)				
P3B3	J2-14	P0.11	PHT0B(I)				
P3B4	J2-15	P0.12	AMP0(I)				
P3B5	J2-16	P0.13	AMP1(I)				
P3B6	J2-17	P0.14	AMP2(I)				
P3B7	J2-18	P0.15	AMP3(I)				

模擬	接頭	泛用	專 用 定 義	模擬	接頭	泛用	專 用 定 義
P2A0	J2-27	P2.0	--	P2C0	J2-43	--	ADC0(I)
P2A1	J2-28	P2.1	--	P2C1	J2-44	--	ADC1(I)
P2A2	J2-29	P2.2	--	P2C2	J2-45	--	ADC2(I)
P2A3	J2-30	P2.3	--	P2C3	J2-46	--	ADC3(I)
P2A4	J2-31	P2.4	--	P2C4	J2-47	--	ADC4(I)
P2A5	J2-32	P2.5	--	P2C5	J2-48	--	ADC5(I)
P2A6	J2-33	P2.6	--	P2C6	J2-49	--	ADC6(I)
P2A7	J2-34	P2.7	--	P2C7	J2-50	--	ADC7(I)
P2B0	J2-35	P2.8	--				
P2B1	J2-36	P2.9	--				
P2B2	J2-37	P2.10	--				
P2B3	J2-38	P2.11	*MCS (O)				
P2B4	J2-39	P2.12	*MA15(O)				
P2B5	J2-40	P2.13	*MA16(O)				
P2B6	J2-41	P2.14	*MA17(O)				
P2B7	J2-42	P2.15	*MA18(O)				

模擬	接頭	泛用	專 用 定 義	模擬	接頭	泛用	專 用 定 義
P0A0	J3-11	P1.0	eeSK(O)	P0B4	J3-19	P1.8	ppCLK(O)
P0A1	J3-12	P1.1	eeCS(O)	P0B5	J3-20	P1.9	ppDIN(O)
P0A2	J3-13	P1.2	eeDI(O)	P0B6	J3-21	P1.10	TxD (O)
P0A3	J3-14	P1.3	eeDO(I)	P0B7	J3-22	P1.11	RxD (I)
P0B0	J3-15	P1.4	ssCK(O)	P0C0	J3-23	P1.12	TxEN (O)*
P0B1	J3-16	P1.5	ssCS(O)	P0C1	J3-24	P1.13	rwLED(O)*
P0B2	J3-17	P1.6	ssDI(O)	P0C2	J3-25	P1.14	RxLED(O)*
P0B3	J3-18	P1.7	ssDO(I)	P0C3	J3-26	P1.15	TxLED(O)*

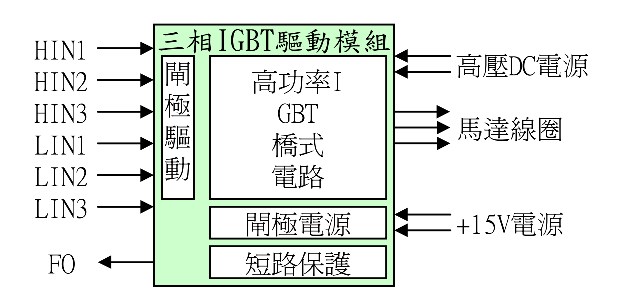
模擬	接頭	泛用	專 用 定 義	模擬	接頭	泛用	專 用 定 義
P1A0	J3-27	--	FPGA擴張點	P1C0	J3-43	--	FPGA擴張點
P1A1	J3-28	--	FPGA擴張點	P1C1	J3-44	--	FPGA擴張點
P1A2	J3-29	--	FPGA擴張點	P1C2	J3-45	--	FPGA擴張點
P1A3	J3-30	--	FPGA擴張點	P1C3	J3-46	--	FPGA擴張點
P1A4	J3-31	--	FPGA擴張點	P1C4	J3-47	--	FPGA擴張點
P1A5	J3-32	--	FPGA擴張點	P1C5	J3-48	--	FPGA擴張點
P1A6	J3-33	--	FPGA擴張點	P1C6	J3-49	--	FPGA擴張點
P1A7	J3-34	--	FPGA擴張點	P1C7	J3-50	--	FPGA擴張點
P1B0	J3-34	--	FPGA擴張點				
P1B1	J3-36	--	FPGA擴張點				
P1B2	J3-37	--	FPGA擴張點				
P1B3	J3-38	--	FPGA擴張點				
P1B4	J3-39	--	FPGA擴張點				
P1B5	J3-49	--	FPGA擴張點				
P1B6	J3-41	--	FPGA擴張點				
P1B7	J3-42	--	FPGA擴張點				

模擬	接頭	泛用	專 用 定 義	模擬	接頭	泛用	專 用 定 義
XD0	MD0	--	MD0	XA0	MA0	--	MA0
XD1	MD1	--	MD1	XA1	MA1	--	MA1
XD2	MD2	--	MD2	XA2	MA2	--	MA2
XD3	MD3	--	MD3	XA3	MA3	--	MA3
XD4	MD4	--	MD4	XA4	MA4	--	MA4
XD5	MD5	--	MD5	XA5	MA5	--	MA5
XD6	MD6	--	MD6	XA6	MA6	--	MA6
XD7	MD7	--	MD7	XA7	MA7	--	MA7
				XA8	MA8	--	MA8
XRD#	MRD	--	MRD	XA9	MA9	--	MA9
XWR#	MWR	--	MWR	XA10	MA10	--	MA10
--	MCS	--	**P2.11	XA11	MA11	--	MA11
				XA12	MA12	--	MA12
XD2	J2-21	--	RESET(I)	XD0	MA13	--	MA13
XD3	J2-22	--	ALARM(I)	XD1	MA14	--	MA14
XD4	J2-23	--	RxUSB(I)	--	MA15	--	**P2.12
XD5	J2-24	--	TxUSB(O)	--	MA16	--	**P2.13
XD6	J2-25	--	RxNET(I)	--	MA17	--	**P2.14
XD7	J2-26	--	TxNET(O)	--	MA18	--	**P2.15

伺服界面部分

功率驅動模組

功率驅動模組可採用Cyntec的IM13100，電路接腳如下：



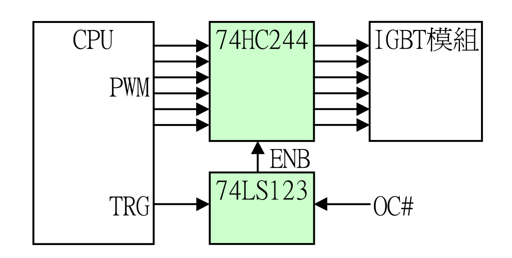
其中：

1. IM13100爲高功率的IGBT驅動模組，工作範圍爲600V/20A，可在將AC110/220V整流濾波後的DC電源下工作。
2. 其內部包括：
   A.高功率IGBT橋式電路，可直接驅動三相交流馬達的三組線圈。
   B.閘極驅動：六組IGBT所需要的閘極驅動電路。
   C.閘極電源：只需+15V的單電源，即可產生高低端所需的四組閘極電源。
   D.短路保護：整個橋式電路對地的短路保護電路。
3. 就數位界面而言，提供TTL型式的邏輯界面，包括：
   HIN1~3：高端的三組閘極控制訊號，1=OFF，0=ON。
   LIN1~3：低端的三組閘極控制訊號，1=OFF，0=ON。
   FO　　：短路保護輸出，0=短路，1=正常。
4. 高低端IGBT的Ton=1.1us而Toff=1.2us，兩者相當對稱，所以1us的切換保護即可安全的處理。
5. 短路保護是在IM13100的內部直接執行，FO訊號只是作短路的指標訊號而已，並不需要DSP端作立即的處理。

WatchDog電路

對於功率驅動電路而言，最危險的狀況就是DSP當機時，整個輸出界面將完全失控(以固定的PWM訊號輸出)。這時必須以硬體電路來切斷所有的輸出界面，以避免功率驅動端產生任何的誤動狀況。

通常我們在輸出端另外用74LS123/74HC244作保護，圖示如下：
其中：

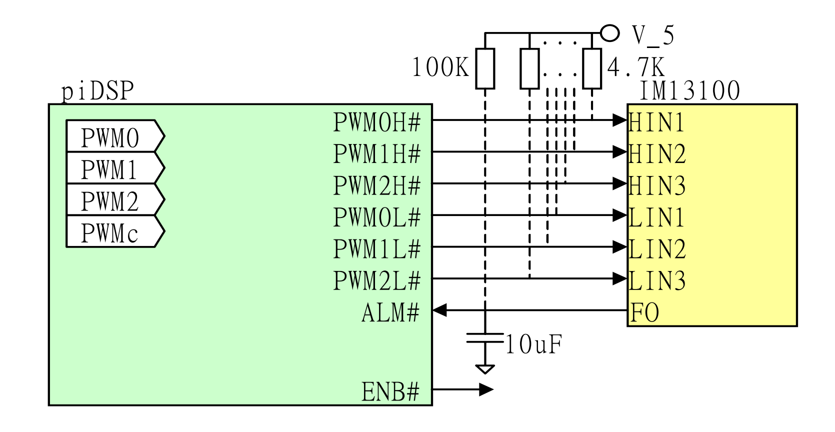


1.CPU所產生的PWM訊號並不能直接接到高功率IGBT模組，而在其中加上74HC244作緩衝。
2.74HC244必須在ENB=0時才准作輸出動作，否則將所有輸出切斷。
3.ENB訊號由74LS123來控制，其作動狀況如下：
  A.當電源開機時產生OC#=0訊號，這時立刻設定ENB=1，切斷所有輸出。
  B.當電源穩定而OC#=1時，才准DSP作控制。
  C.當DSP開機完成並穩定的執行程式時，會從TRG端固定送出1KHz的脈衝訊號。
  D.這時74LS123會以One-Shoot方式的單穩態電路設定ENB=0，允許PWM輸出。
  E.一旦CPU出狀況而不再送出TRG訊號時，這時不論TRG=0或1，都會在1ms內設定ENB=1而切斷所有輸出。

所以TRG的脈衝訊號不得用硬體自動產生，一定要由程式的執行來產生。如此當軟體程式當機時，才會正確的標示當機狀況而立刻處理。

PWM驅動界面

piDSP中內建了WatchDog電路，簡化了PWM驅動界面。圖示如下：



其中：

1.DSP的六組PWM輸出訊號可直接連接IGBT功率模組(例如IM13100)，只要外加pull-up電阻即可。

2.DSP中內建WatchDog保護電路，包括：

A.當電源開機或輸出短路時，產生ALM#=0訊號，這時立刻切斷所有PWM輸出。

B.當電源穩定而輸出沒有短路時(ALM#=1)，才准DSP作PWM的輸出控制。

C.DSP程式中必須以1KHz以上的頻率觸發PWMc中的TRG參數，才准PWM輸出。

D.一旦DSP出狀況而不再送出TRG參數時，會在1ms內切斷所有PWM輸出。

E.ENB輸出訊號用以控制其他需要保護的外加界面電路，例如FPGA。

就軟體界面而言，PWM驅動共佔用4-word的I/O界面，定義如下：

1.PWM0/PWM1/PWM2用來設定三組PWM的作動時間(12-bit)，範圍爲0~4095。

2.PWMc爲16-bit的PWM命令，其中包括：

b15　：參數模式，0=TRG設定模式，1=PWM參數設定模式。

b11~8：FRQ參數(4-bit)，用來設定PWM頻率。

b5~1　： DZ參數(5-bit)，用來設定PWM輸出的切換延遲。

b0　　：TRG參數(1-bit)，以程式設定的方波訊號。

3.FRQ參數用來設定PWM的頻率，可以選擇FRQ=1~15，計算公式爲

PWM頻率 = FRQ * 1.5KHz

所以PWM的頻率範圍爲1.5KHz(FRQ=1)至22.5KHz(FRQ=15)。

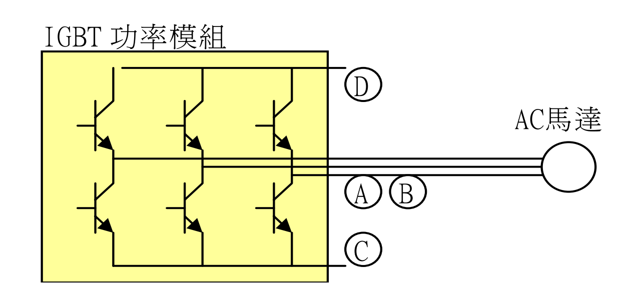
4.DZ參數用來設定PWM切換保護上的Dead-Zone，可以選擇DZ=0~31，計算公式爲

Dead-Zone = DZ * 0.16us

所以切換保護的範圍爲0us(DZ=0)至5us(DZ=31)。

電流電壓偵測

在三相交流馬達的控制上，電流和電壓偵測是很重要的一環。圖示如下：



其中：

1.電流和電壓的偵測點包括：

#A/#B：三相交流馬達的各相電流。由於三相電流的總和爲零，所以只要偵測其中的兩相電流，即可換算取得第三相的電流。

#C　：電源電流，可以監視負載和短路狀態。

#D　：電源電壓，可以監視電壓過高或過低的狀況，也可以依照電源電壓來調整控制器的內部增益。

2.由於#A/#B兩點是浮動的，所以必須用特殊的電流偵測器，可以選擇：

霍爾元件：其輸出是隔離式的類比電壓，可以用ADC直接讀入。缺點是需要+/-15V的額外電源。

IR2172S　：其輸出是隔離式的數位邏輯，由於是PWM型式的訊號，必須另外以特殊界面作處理。

3.#C的偵測非常單純，只是偵測串聯電阻端的電壓而已，可直接用ADC處理。

4.#D的偵測也非常簡單，只要用降壓電阻將高壓電源降低到ADC的處理範圍，即可由ADC直接處理。

因此電壓和電流的偵測有兩種選擇：

1.類比式的電流偵測，需要四組ADC頻道。

2.數位式的電流偵測，需要兩組數位偵測和兩組ADC頻道。

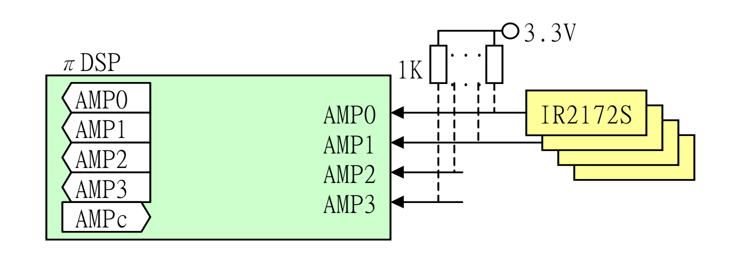
IR2172S的PWM輸出訊號的特徵如下：

1.IR2172S的PWM輸出訊號，其頻率固定在40KHz。但由於工作環境的溫度問題，其頻率會有+/-10%左右的飄移。

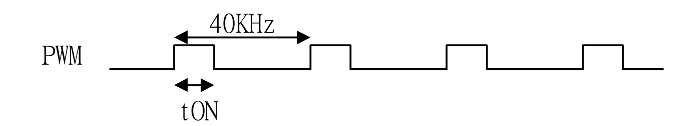
2.所量測的電流値對映到PWM訊號中的負載率，只要偵測負載率即可換算出對映的電流値。

PWM輸入界面

piDSP中內建了PWM輸入處理界面，簡化了電流偵測的處理。圖示如下：



而PWM訊號的偵測方式如下：

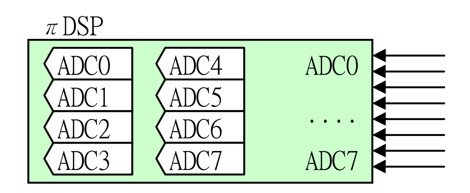


其中：

1.偵測PWM訊號中的tON週期，存入I/O界面區所對映的AMPn中。
2.偵測PWM訊號的計數值，並存入I/O界面區所對映的CNTn中。
3.如果40KHz的頻率不變的，則計算(tON*40KHz)即可對映PWM訊號的負載率。
4.如果40KHz的頻率會隨溫度作飄移，則可在固定的週期下對CNTn作差分計算，取得頻率的對映值，即可計算(tON*FRQ)而換算出電流值。
5.所以在I/O界面區共需要4-word的空間，分別爲AMP0~3。其中：
   b9~0　：對映到四組PWM訊號的tON值，在24.576MHz的系統頻率下，其偵測範圍在0~614之間，相當於9-bit的精度。
   b15~11：對映到四組PWM訊號的計數值，若以1KHz爲抽樣單位，40KHz的計數值將需要6-bit來處理。

ADC輸入界面

piDSP中內建八組ADC輸入界面，圖示如下：



其中：

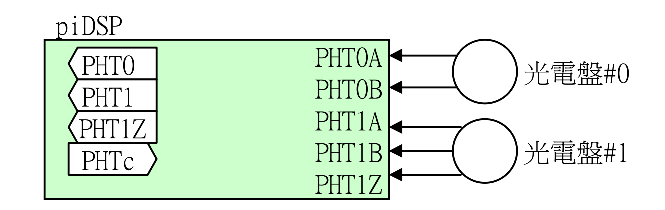
1.ADC輸入共有8組，都是10-bit的精度，訊號範圍爲0~3.3V。

2.在I/O界面區共需要8-word的空間，分別爲：

ADC0~7：直接對映8組訊號的ADC轉換結果。

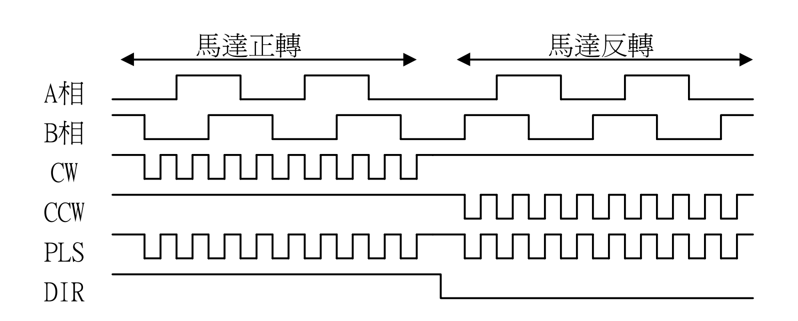
光電寫碼器界面

piDSP中的光電寫碼器界面可外接兩組光電盤，圖示如下：



其中：

1.所以需要兩組光電盤，是基於下述的需求：
   A.在高精度的應用場合，可能同時需要光電盤和光學尺的回授。
   B.在傳統的伺服界面中，位置命令通常也是以A/B相的型式作輸入命令。
2.光電盤解碼結果存在I/O界面區中的PHT0、PHT1和PHT1Z，精度爲12-bit。其中
   PHT0/PHT1：光電盤解碼計數結果，不受Z相的影響。
   PHT1Z　　：光電盤解碼計數結果，當Z相產生時會作歸零處理。
3.光電盤的輸入模式有四種，由PHTc中的4-bit作選擇，包括：
   b7~6：PHT1的模式選擇，0=A/B相，1=CW/CCW，2=DIR/PLS，3=多工?。
   b5~4：PHT0的模式選擇，0=A/B相，1=CW/CCW，2=DIR/PLS，3=多工?。

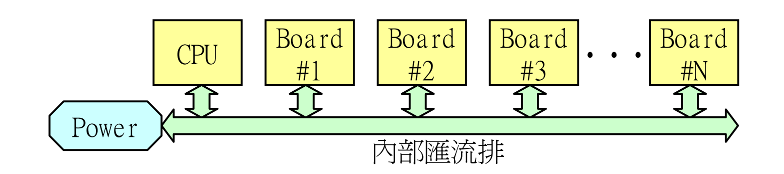


4.四種模式的波形如下：

光纖界面部分

內部連線型光纖網路

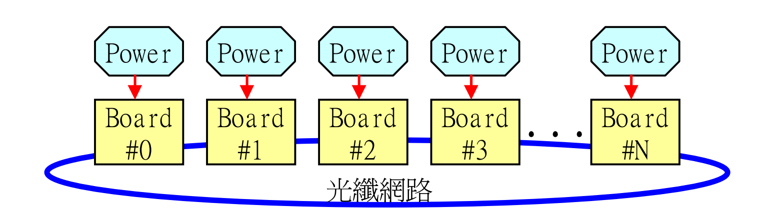
一般電腦中的匯流排架構如下：



其中：

1.所有模組共用DC電源，訊號是共地(Common Ground)的型式。
2.CPU和所有I/O模組之間採用緊密型連結，可由CPU來讀寫各模組中的記憶體和暫存器。
3.同常採用並列型連線，抗雜訊能力有限，所以傳訊距離非常短，約30cm左右。

piDSP中光纖網路的規劃目標，就是取代這些內部匯流排，結構如下：



其中：

1.各個模組的DC電源可以完全獨立，不必採取共地的型式。
2.各個模組之間也是採用緊密型連接，每個模組都可讀寫其他模組中的記憶體和暫存器。
3.藉著光纖網路的抗雜訊能力，可以將傳訊距離拉開至10m左右，並能穩定工作於高雜訊的工作環境。

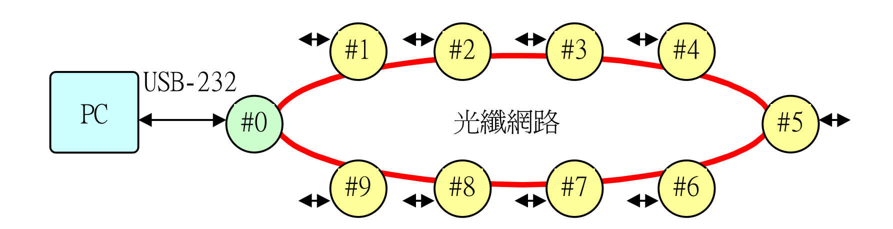
所以在piDSP的規劃中，光纖網路所提供的只是另一種內部匯流排，將各個piDSP連線而能相互的傳遞記憶體和暫存器資料。其設計重點在獨立而隔離的電源系統和高度的抗雜訊能力，和一般適用於長距離通訊的光纖網路是完全不同的。

USB232的處理

光纖網路採用主從式的節點架構，其特性包括：

1. 網路節點最多可接30組，以5-bit的節點編號標示，範圍爲0~31，定義如下：
   - #0　　：保留給主節點。
   - #1~#30：最多30組的輔節點。
   - #31　：保留給以網路序號來取代節點編號的節點選擇方式。
2. 主節點中的TxNET端是獨立控制的，不受RxNET端所影響。而輔節點中的TxNET端是跟隨RxNET端作同步控制的。
3. 不論主節點或輔節點都可作DROP/PASS/ADD的處理，這方面的功能並無差別。

由於光纖網路只是各節點之間的內部連線，必須另外藉著USB-232界面作外部的連線工作，圖示如下：



其中：

1. USB-232是光纖網路所指定的外部連線方式，每個piDSP元件都具備這種界面。
2. USB-232是利用USB界面來傳遞的RS232訊號，其傳訊速度固定爲1.5Mbit/sec。
3. 從PC端而言這是標準的USB界面，而從piDSP端而言這只是高速RS232界面。
4. USB-232所執行的是非同步的通訊連線，在piDSP中會自動轉換成光纖訊號後傳遞至每一個網路節點中。
5. 換句話說，雖然每個piDSP中都有USB-232界面，但是只准有一組能真正連接到PC，其他所有節點的USB-232界面都必須空置。
6. 若連接到PC的是主節點，則PC不但能透過USB-232來讀寫主節點，也可再透過光纖網路來讀寫所有的網路節點。
7. 若連接到PC的是輔節點，則PC只能透過USB-232來讀寫該節點，而不能讀寫其他的網路節點。
8. 換句話說，只有主節點具有USB-232轉換成光纖網路的傳遞能力，其他輔節點都不具備這項功能。

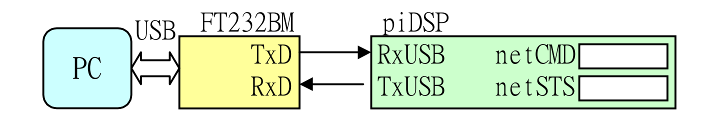
當PC端要讀寫各個網路節點時，只要定義

5-bit的節點編號、15-bit的位址定義、16-bit的資料、和1-bit的讀寫命令

就能順利讀寫指定節點32K-word記憶體中的任何位址。

USB-232界面

USB-232的界面定義如下：



其中：

1.就硬體而言，USB-232界面佔用DSP元件中的2個接腳，包括：
RxUSB：USB-232界面的接收端，通訊格式爲1.5M/8/E/2。
TxUSB：USB-232界面的發射端，通訊格式爲1.5M/8/E/1。

2.外部元件FT232BM負責處理USB界面和RS232界面的轉換工作，在轉換成USB界面後即可連線至PC端。

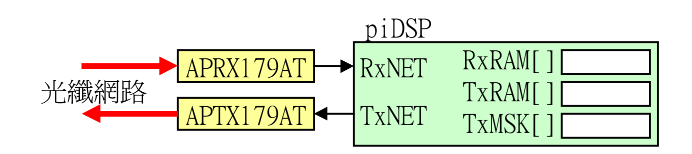
3.就軟體而言，USB-232佔用兩組I/O界面，定義如下：

I/O界面	讀寫程序	位　元　定　義
netCMD	寫入0x5F	b15~8：8-bit的block長度，範圍爲1~128 b7~6 ：暫時保留 b5 ：1=主節點，0=輔節點 b4~0 ：節點編號，範圍爲0~31
netSTS	讀取0x6F	b15~8：RxNET端所接收正常word的8-bit計數器 b7~0 ：RxNET端所接收錯誤word的8-bit計數器

4.換句話說，USB-232界面所執行的工作是
由PC端來讀寫指定網路節點32K-word雙阜記憶體的任何位址
就piDSP端的程式設計者而言，除了設定網路編號外，不再需要作任何處理。

光纖網路界面

光纖網路的界面定義如下：



其中：

1.就硬體而言，光纖網路界面佔用DSP元件中的2個接腳，包括：
   RxNET：直接連接光纖網路的接收元件(APRX179AT)。
   TxNET：直接連接光纖網路的發射元件(APTX179AT)。

2.就軟體而言，光纖網路界面佔用的記憶體空間定義如下：

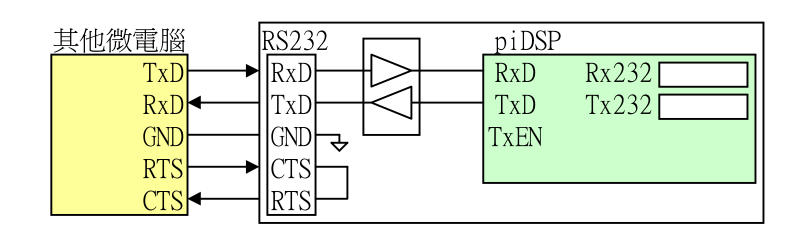
I/O界面	讀寫程序	位　元　定　義
RxRAM[128]	0x0100~ 0x017F	直接對映從RxNET端所接收的各個word
TxRAM[128]	0x0180~ 0x01FF	直接對映從TxNET端所發射的各個word
TxMSK[8]	0x0078~ 0x007F	直接對映TxRAM[ ]中各個word的控制選擇 TxMSK=0：PASS處理 TxMSK=1：ADD 處理 其中 TxMSK[0]：對映word#0 ~word#15的選擇 TxMSK[1]：對映word#16~word#31的選擇 .... TxMSK[7]：對映word#112~word#127的選擇

3.只有RxRAM[0]和TxRAM[0]的定義特殊，對映到word#0的處理。其他各個word的處理都是標準的程序。

UART界面

RS232/422/485界面

Rs232界面連線的規劃如下：



其中：

1.就硬體而言，RS232界面佔用DSP中的3根接腳，包括：
   RxD ：RS232的串列輸入訊號。
   TxD ：RS232的串列輸出訊號。
   TxEN：RS232的串列輸出作動訊號，當TxD輸出作動時自動設定TxEN=1。
2.就軟體程式而言，RS232界面佔用兩組記憶體，包括：
   Rx232：RS232的輸入資料和目前狀態。
   Tx232：RS232的輸出資料和設定命令。
3.Tx232參數定義有兩種，當bit15=1時爲命令設定，而bit15=0時爲輸出資料。
4.在命令設定時，Tx232的bit15固定爲1，而其他bit的定義如下：
   bit14 ：INH，0=NOP，1=當TxEN=1時取消RxD的解碼工作。
   bit13 ：輸出反向處理，0=輸出正向，1=輸出反向。
   bit12 ：輸入反向處理，0=輸入正向，1=輸入反向。
   bit11~10：PAR，0/1=NOP, 2=EVEN, 3=ODD
   bit9~8 ：LEN定義RxD的訊號字長，可選擇7/8/9三種。
      0=7-bit，1=8-bit，2=9-bit。
   bit7~0 ：PRD(0~255)，用來設定RS232的Baud-Rate，其計算公式爲
      Baud-Rate = 24576000 /PRD /16
5.在輸出資料時，，Tx232的bit15固定爲0，而其他bit的定義如下：
   bit14 ：TxCMD，當TxCMD和TxSTS不同時代表要送出TxD資料。
   bit8~0 ：9-bit的輸出資料。
6.在讀取資料時，Rx232的定義如下：
   bit15 ：RxSTS，當RxSTS變號時代表讀入新資料
   bit14 ：TxSTS，當TxCMD和TxSTS相同時代表TxD可以接受新資料。
   bit9 ：對映到bit7~0的Parity檢驗結果，0=ok,1=error。
   bit8~0 ：9-bit的輸入資料。

通訊規格

基本上，piDSP中的RS232界面參考8051的設計，可同時容許標準RS232界面和不甚標準的8051界面。

當piDSP在執行TxD輸出時，其波形定義如下：

```
TxD   |D0|D1|D2|D3|D4|D5|D6|PAR|         LEN=7-bit
TxD   |D0|D1|D2|D3|D4|D5|D6|D7|PAR|      LEN=8-bit
TxD   |D0|D1|D2|D3|D4|D5|D6|D7|D8|       LEN=9-bit
TxEN  0  1  2  3  4  5  6  7  8  9  10  11  12
```

其中：

1. TxD的輸出長度固定為12-bit，包括
   - STT：1-bit的Start-bit，固定為0
   - DAT：指定長度的Data，可選擇7/8/9-bit三種。
   - PAR：1-bit的Parity-bit，可選擇Non/Even/Odd三種。
   - STP：2-bit的Stop-bit，固定為1。
2. 基本上9-bit的DAT直接對映到Tx232中的bit8~0，當PAR選擇Even或Odd時，自動再增加1-bit，否則在DAT碼後直接送出STP碼。
3. 當字長選擇9-bit時，PAR碼必須選擇Non，這時整個9-bit都可自由設定。
4. 所以在固定12-bit的長度下，可選擇字長為7/8-bit的標準RS232規格，也可選擇字長為9-bit的特殊規格。
5. 字長為9-bit的UART結構非常適合簡單的8-bit參數設定工作，例如：
   - bit8=1時為命令模式，這時bit7~0可用來定義讀寫命令和位址。
   - bit8=0時為資料模式，這時bit7~0直接對映8-bit的讀寫資料。

而從RxD端讀回資料時，其解碼程序如下：

1. 固定將RxD中的D0~D8(9-bit)，讀入Rx232中的bit8~bit0。
2. Rx232中的bit9固定放置D0~D7(8-bit)的Parity檢驗結果，1=error,0=ok。
3. 由Tx232中的LEN參數決定正確的RxD解碼長度。

由於piDSP採用雙阜記憶體的I/O界面定義，程式設計者並不能直接控制I/O界面，而只能讀寫對映到I/O界面的記憶體，再由系統自動以32KHz的頻率更新所有的I/O界面。因此：

1. 當寫入Tx232參數時，並不能馬上寫入RS232界面。
2. 當讀取Rx232資料時，只是讀取上次32KHz的更新結果。
3. 即使沒有任何讀寫記憶體的動作，還是會以32KHz的頻率更新所有I/O界面。

因此正確的UART控制程序如下：

1. 設定DSP的即時控制頻率，
   - 當Baud= 9600時，只要DSP的控制頻率設在1KHz以上即可。

當Baud=19200時，DSP的控制頻率必須設在2KHz以上，依此類推。

2.所有UART的處理程式必須放在@timer程序中，而以定時中斷的方式作處理。

3.首先設定通訊規格，可接設定Tx232參數。在延遲一個@timer週期後才能繼續作其他的讀寫處理。

4.TxD的處理程序如下：

A.讀取Tx232和Rx232，確定TxCMD和TxSTS相同。這是TxD準備好的狀況。

B.若TxD準備好，則設定Tx232並將其中的TxCMD變號，這是下達TxD輸出命令。

C.同一個@timer週期中不得再做其他的TxD輸出控制。

D.等待下一個@timer週期，再回至步驟A送出下一碼。

5.RxD的處理程序如下：

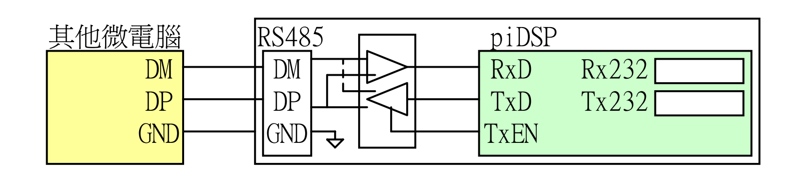
A.讀取Rx232，如果RxSTS變號，就是有新資料從RxD送入的狀況。

B.這時可根據Rx232中的bit9~0來確定資料和資料的正確性。

C.同一個@timer週期中不得再做其他的RxD輸入控制。

D.等待下一個@timer週期，再回至步驟A檢查下一碼的輸入狀況。

當應用到RS422界面時，只要變換外部的驅動界面即可。而當應用到RS485界面時，其應用電路如下：

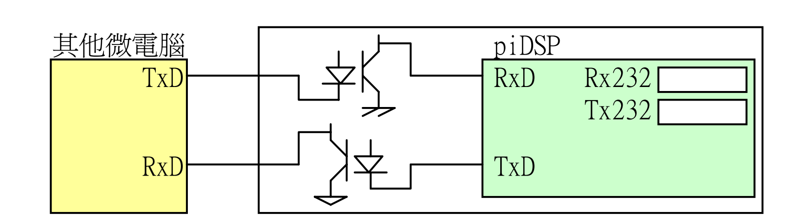


其中：

1.利用TxEN來執行TxD的輸出控制，以建立雙向的通訊功能。

2.可以設定Tx232中的INH參數，當TxD輸出資料時就取消RxD的讀入解碼動作，可避免額外的軟體解碼負擔。

當需要光耦合隔離時，應用電路如下：



其中：

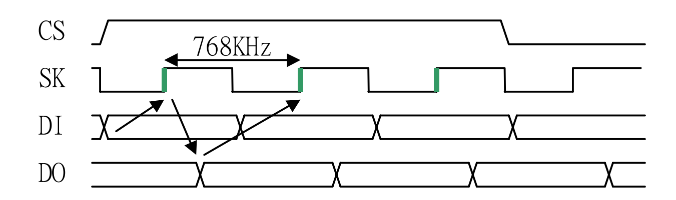
1.利用光耦合隔離來執行電源隔離的工作，使得各微電腦之間不用共地。

2.許多光耦合元件具有反向的效應，這時可設定輸入和輸出的反向功能作補償。

EEPROM界面部分

AT93C46/56/66/86

AT93C46採用串列界面。波形格式如下：



其中：

1.界面訊號包括：
  CS　　　：Chip Select
  SK/DI/DO：Serial Data Clock/Input/Output
2.word格式是以CS=1開始，而以CS=0結束。bit格式是在SK的上緣觸發。
3.預設的波形規劃如下：
  A.SK是768KHz的方波，週期約爲1.3us，符合1MHz的頻率上限。
  B.CS和DI訊號都在SK的下緣更新。
  C.DO可在下一個SK的上緣作讀入。
4.記憶體格式可選擇16-bit或是8-bit，但piDSP中固定以8-bit格式處理。

讀寫指令的格式表列如下：

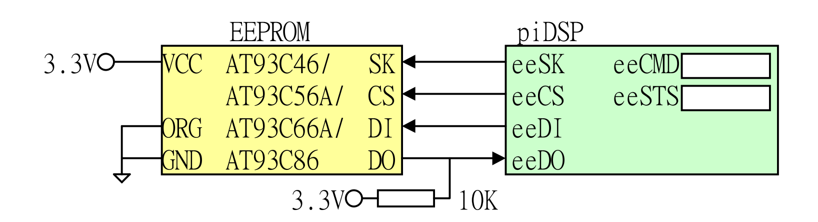
指令	bit	DI格式	DO格式	說明
READ	18	110+A[7]	D[8]	讀入16-bit資料
EWEN	10	10011+X[5]	--	Erase/Write Enable指令
ERASE	10	111+A[7]	--	清除16-bit資料
WRITE	18	101+A[7]+D[8]	--	寫入16-bit資料
ERAL	10	10010+X[5]		Erase All指令
EWDS	10	10000+X[5]		Erase/Write Disable指令

其中：

1.讀取動作可在一個讀寫週期後結束，但是寫入動作需要再等10ms。
2.AT93C46/56/66的讀寫指令都類似，只是容量不同，其中
  AT93C46：容量爲 128*8，其位址以A[7]作設定。
  AT93C56：容量爲 256*8，其位址以A[8]作設定。
  AT93C66：容量爲 512*8，其位址以A[9]作設定。
  AT93C86：容量爲2048*8，其位址以A[11]作設定。

EEPROM界面

piDSP和EEPROM的連線界面規劃如下：

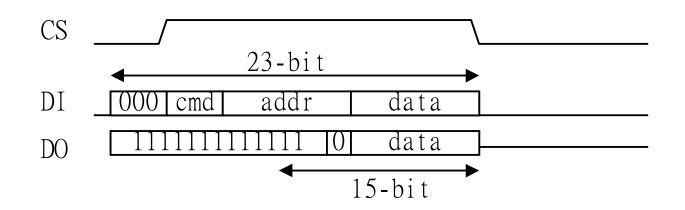


其中：

1.就硬體而言，EEPROM界面佔用DSP元件中的4個接腳，包括：
  eeSK：直接連接至EEPROM中的SK。
  eeCS：直接連接至EEPROM中的CS。
  eeDI：直接連接至EEPROM中的DI。
  eeDO：直接連接至EEPROM中的DO。
2.EEPROM可以選擇
  AT93C46　：　1Kbit的EEPROM，對映到 128-byte。
  AT93C56A：　2Kbit的EEPROM，對映到 256-byte。
  AT93C66A：　4Kbit的EEPROM，對映到 512-byte。
  AT93C86　：16Kbit的EEPROM，對映到2048-byte。
3.EEPROM中的其他接腳包括：
  VCC：直接連接+3.3V電源
  GND：直接連接地線
  ORG：連接GND，即可選擇8-bit的讀寫模式。
4.EEPROM的DO以10KΩ電阻作pull-up，所以當DO浮動時從DSP端可讀到邏輯1。
5.就軟體而言，EEPROM佔用兩組I/O界面，但相當於三組暫存器，定義如下：

I/O界面	讀寫程序	位　元　定　義
eeCMD	寫入0x18 且b15=1	b15　：固定為1 b14~0：15-bit的(0+命令+位址)
eeCMD	寫入0x18 且b15=0	b15　：固定為0 b7~0　：8-bit的寫入資料
eeSTS	讀取0x19	b15　：STS=1正在執行，STS=0執行完畢 b14~0：15-bit的讀入資料， 通常只有最低的8位元有效

不論EEPROM的容量為何，讀寫格式固定如下：



其中：

1.不論是任何指令，DI的寫入固定爲23-bit而DO的讀取固定爲15-bit。

2.若要在Addr=0x01的位址寫入0x55時，不同EEPROM的寫入格式如下：

AT93C46 ：eCMD的位址和命令=(1+00000101+0000001)
AT93C56A：eCMD的位址和命令=(1+0000101+00000001)
AT93C66A：eCMD的位址和命令=(1+000101+000000001)
AT93C86 ：eCMD的位址和命令=(1+0101+00000000001)
而eCMD的資料固定爲 (0+0000000+01010101)

3.換句話說，在23-bit的DI訊號中，格式固定爲15-bit的(cmd+addr)和8-bit的data。當(cmd+addr)不滿15-bit時，不足的部分在前方以0塡滿。

4.由於EEPROM命令的起動點固定爲邏輯1，piDSP將自動忽略前置的邏輯0部分，而產生上圖中的CS訊號。相當於自動的調整DI訊號總長。

5.同樣在DO訊號讀取時，不只是讀取最後的8-bit，而是讀取最後的15-bit資料作儲存。多餘的資料可用來自動偵測EEPROM的容量。

6.在768KHz的CK頻率下，EEPROM的讀寫動作固定爲768KHz/23=33KHz。

7.由於piDSP的中斷頻率最高爲32KHz，所以在一個@timer週期中中一定可以完成EEPROM的讀寫動作。

EEPROM應用範例

讀取控制程序：

1.設定eeCMD的位址和命令，再寫入eeCMD的資料(通常爲0)。
2.檢查eeSTS中的b15，當b15=0時即是EEPROM已經準備完畢。
3.這時eeSTS中的b7~0就是要讀取的EEPROM資料。

寫入控制程序：

1.設定eeCMD的位址和命令，再寫入eeCMD的資料。
2.檢查eeSTS中的b15，當b15=0時即是EEPROM已經寫入完畢。

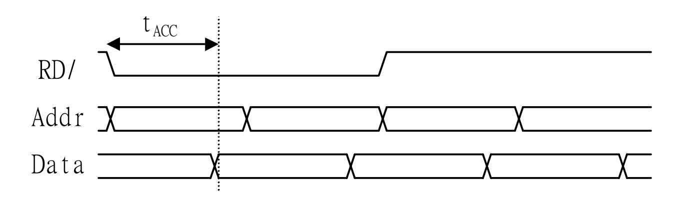
自動辨識程序：

1.設定eeCMD的位址和命令，再寫入eeCMD的資料。其中
 位址命令固定爲(1+0110+00000000000)，即是讀取位址#0的指令。
 寫入資料固定爲(0+0000000+00000000)，是無意義的資料。
2.檢查eeSTS中的b15，當b15=0時即是EEPROM已經準備完畢。
3.這時eeSTS中的b14~0就是所讀取的內容。其中
 若b14~0=(110xxxx+xxxxxxxx)，則EEPROM爲AT93C46。
 若b14~0=(1110xxx+xxxxxxxx)，則EEPROM爲AT93C56A。
 若b14~0=(11110xx+xxxxxxxx)，則EEPROM爲AT93C66A。
 若b14~0=(1111110+xxxxxxxx)，則EEPROM爲AT93C86。
4.這是因爲在位址和命令尚未送完時，EEPROM的DO端會呈現浮動狀態，由於10K的pull-up電阻處理，DSP端會讀入邏輯1。
5.一旦位址和命令送完而開始讀入資料時，EEPROM的DO端會先送出一個邏輯0後才繼續送出8-bit資料。
6.因此偵測邏輯1/0的交界處，就可以確認(位址和命令)的長度，也因而確認EEPROM的型號和容量。
7.偵測出型號和容量後，才可以根據不同的型號而下達正確的(位址和命令)值。

FlashROM界面部分

AT29C010A/W29FE011

AT29C010A採用並列界面。其資料讀取的波形格式如下：



其中：

1.界面訊號包括：

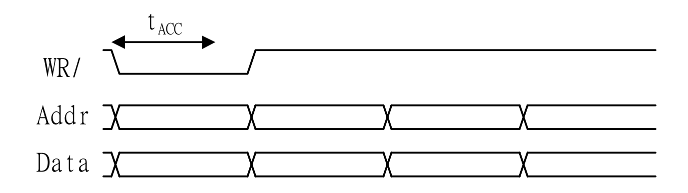
CS/ ：Chip Select，可以固定設爲0，即是永遠作動。
RD/ ：Read Enable，0=讀取，1=NOP。
WR/ ：Write Enable，0=寫入，1=NOP。
Addr：位址匯流排，爲17-bit。
Data：資料匯流排，爲 8-bit。

2.每個byte在讀取時，從位址設定到資料送出的延遲時間爲t_{ACC}，就是FlashROM規格中的Access Time，可選擇70/90/120ns等不同規格。

3.以24.576MHz的系統時鐘而言，

在t_{ACC}= 70ns時，必須延遲2個系統時鐘才能正確讀入資料。
在t_{ACC}= 90ns時，必須延遲3個系統時鐘才能正確讀入資料。
在t_{ACC}=120ns時，必須延遲3個系統時鐘才能正確讀入資料。

而命令寫入的波形格式如下：



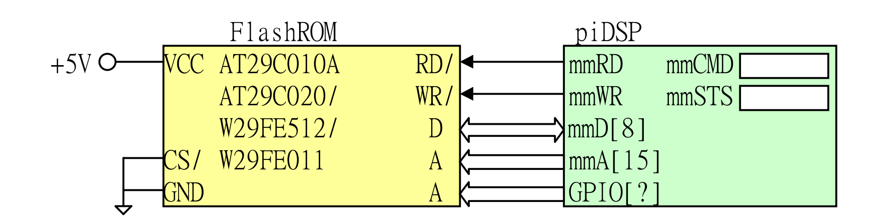
其中：

1.整個寫入週期和t_{ACC}相當。

2.所謂寫入動作，只是命令設定而已。FlashROM以page(128-byte)爲寫入單位，每個page的寫入都需要一連串的指令才能完成,而在page寫入後還需等待10ms才能執行下一個動作。

FlashROM界面

DSP和FlashROM的連線界面規劃如下：



其中：

1.就硬體而言，FlashROM界面佔用DSP元件中的25個接腳，包括：

mmRD　：直接連接至flashROM中的RD/。

mmWR　：直接連接至flashROM中的WR/。

mmD[8]：直接連接至flashROM中的D[8]。

mmA[15]：直接連接至flashROM中的A[15]。

2.當FlashROM的容量>32Kbyte時，可用GPIO界面來設定位址匯流排中的高位元。

2.FlashROM可以選擇

AT29C256　　　　　：256Kbit(32K-byte)的FlashROM，位址爲15-bit。

W29FE512A　　　　　：512Kbit(64K-byte)的FlashROM，位址爲16-bit。

AT29C010A/W29FE011A：　1Mbit(128K-byte)的FlashROM，位址爲17-bit。

AT29C020 /W29FE020C：　2Mbit(256K-byte)的FlashROM，位址爲18-bit。

AT29C040A/W29FE040 ：　4Mbit(512K-byte)的FlashROM，位址爲19-bit。

3.FlashROM中的其他接腳包括：

VCC：直接連接+5V電源

GND：直接連接地線

CS/：連接GND，即固定連接單個FlashROM，不作其他選擇。

4.就軟體而言，EEPROM佔用兩組I/O界面，但相當於三組暫存器，定義如下：

I/O界面	讀寫程序	位　元　定　義
mmCMD	寫入0x20 且b15=1	b15　：固定爲1 b14~0：15-bit的位址
mmCMD	寫入0x20 且b15=0	b15　：固定爲0 b7~0　：8-bit的寫入資料
mmSTS	讀取0x21	b7~0　：8-bit的讀取資料，

5.要從EEPROM中讀取資料時(以8-bit爲單位)，程序如下：

A.以mmCMD(且b15=1)來設定讀取位址，設定一次後就不需要再變更。

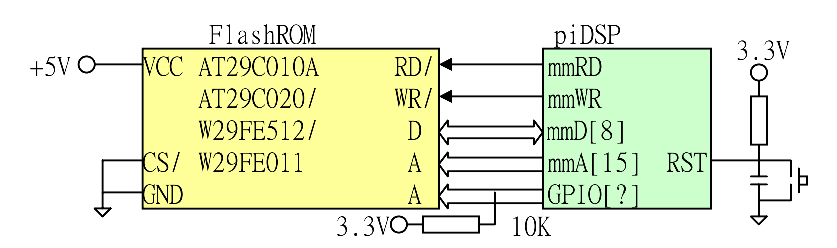
B.在4個DCLK週期後，讀取mmSTS。第一筆資料不要處理。

C.每4個DCLK週期可讀取mmSTS一次，將8-bit資料依序讀入。

6.要對FlashROM中寫入資料時(以8-bit爲單位)，先以mmCMD(且b15=0)來寫入資料，再以mmCMD(且b15=1)來寫入位址。重複循環即可寫入多筆資料。

開機下載程序

DSP元件的開機啟動界面如下：



其中：

1.電源開機或是系統復歸時，送入RST訊號而進入開機狀態。

2.開機狀態下所有I/O界面復歸成輸入模式，所以GPIO[?]接上pull-up電阻後，其開機時的預設值將是邏輯1。

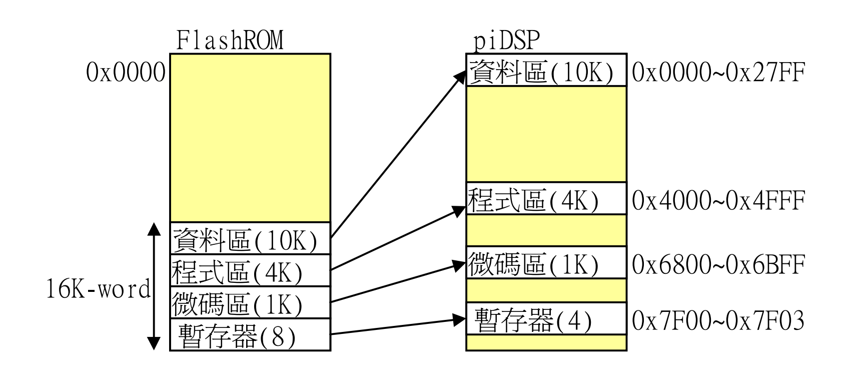
3.這時開始自動下載程序，包括：

A.mmA[15]從0x0000~0x7FFF，自動掃瞄所有的32K-byte區域，從FlashROM讀出後即設定RAM的初值，包括資料區、程式區和暫存器。

B.最後自動啓動DSP程式，開始執行DSP程式>

4.所有其他的開機設定程序都必須由DSP程式中自行處理，包括FPGA電路下載。

FlashROM中的空間分配狀況如下：



其中：

1.開機程式佔用FlashROM中最後的16K-word的空間，包括

資料區：對映到16K-word資料區中的10K-word。

程式區：對映到4K-word程式區的全部。

微碼區：對映到1K-word微碼區的全部。

暫存器：將4-word暫存器重複寫入兩次。

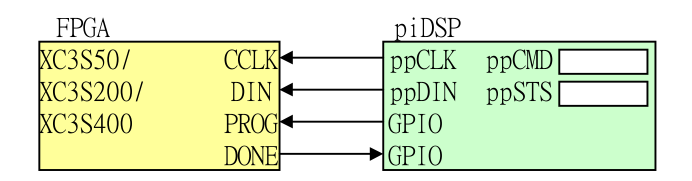
2.對於0x00~0x1FF的I/O界面區，不論FlashROM內容都固定的下載0x0000。

3.若FlashROM的byte#2~3的內容爲0x5AA5，則自動下載FlashROM的內容，否則將下載0x0000至所有的DSP下載區域。

FPGA界面部分

FPGA界面

FPGA界面包括下載界面和通訊界面兩部分。下載界面圖示如下：



其中：

1.就硬體而言，FPGA下載界面佔用DSP元件中的4個接腳，包括：
ppCLK　：直接連接至FPGA中的CCLK，頻率約爲1MHz。
ppDIN　：直接連接至FPGA中的DIN。
GPIO[2]：以GPIO界面直接連接到FPGA中的PROG和DONE。

2.FPGA可選擇XC3S50、XC3S200或XC3S400，閘數分別爲50K、200K和400K。
XC3S50　：電路佔用439K-bit，約爲 28K-word的空間。
XC3S200：電路佔用　1M-bit，約爲 66K-word的空間。
XC3S400：電路佔用1.7M-bit，約爲106K-word的空間。

3.FPGA的下載，基本上是串列處理電路，包括：
PROG：啓動下載程序。
CCLK：串列傳輸的CLK部分。
DIN　：串列傳輸的DATA部分。

4.就軟體而言，FPGA下載界面佔用兩組I/O界面空間：

I/O界面	讀寫程序	位　元　定　義
ppCMD	寫入0x1C	b15~0：16-bit的下載資料
ppSTS	讀取0x1D	b15　：STS=1正在執行，STS=0執行完畢

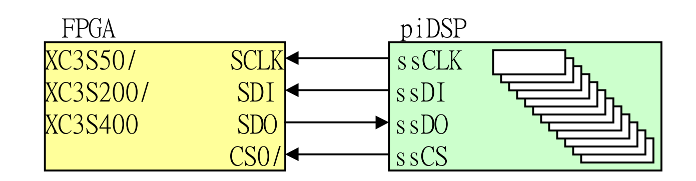
5.要對FPGA作下載處理時，程序如下：
A.以GPIO設定PROG=0，等待一段時間後再設定PROG=1。
B.從FlashROM中讀出1-word後，設定ppCMD暫存器，即會依序作串列傳輸。
C.每個32KHz的抽樣週期後重複執行步驟#B，直到所有電路下載完畢。

下面是相關FPGA所需要的下載時間：

抽樣頻率	XC3S50	XC3S200	XC3S400
DSP程式處理	0.5秒	1秒	2秒
光纖網路下載	1秒	2秒	4秒

通訊界面

FPGA的通訊界面則依照固定的SIO界面結構，圖示如下：



其中：

1.就硬體而言，FPGA通訊界面佔用DSP元件中的4個接腳，包括：
   ssCLK：直接連接至FPGA中的SCLK。
   ssDI ：直接連接至FPGA中的SDI。
   ssDO ：直接連接至FPGA中的SDO。
   ssCS ：直接連接至FPGA中的CS0/。
2.串列傳輸的傳輸速度是12.288Mbit/sec，對映到
   8組12-bit的輸入阜， 8組12-bit的輸出阜，
   16組16-bit的輸入阜，16組16-bit的輸出阜。
   相當於352-bit的輸入阜和352-bit的輸出阜。
3.所有輸入阜和輸出阜都對映到RAM的I/O界面區，而以32KHz抽樣頻率作更新。
4.I/O界面的對映表格如下：

位址	CHN=0	CHN=1	CHN=2	CHN=3	CHN=4	CHN=5	CHN=6	CHN=7
0x20~0x27	IPB0	IPB1	IPB2	IPB3	IPB4	IPB5	IPB6	IPB7
0x30~0x37	OPA0	OPA1	OPA2	OPA3	OPA4	OPA5	OPA6	OPA7
0x40~0x47	IPC0	IPC1	IPC2	IPC3	IPC4	IPC5	IPC6	IPC7
0x50~0x57	OPB0	OPB1	OPB2	OPB3	OPB4	OPB5	OPB6	OPB7
0x60~0x67	IPA0	IPA1	IPA2	IPA3	IPA4	IPA5	IPA6	IPA7
0x70~0x77	OPC0	OPC1	OPC2	OPC3	OPC4	OPC5	OPC6	OPC7

其中IPA0~7爲8組的12-bit輸入阜，OPA0~7爲8組的12-bit輸入阜，而其他輸入輸出阜都爲16-bit。

FPGA模擬器連線

功能分類	piDSP	DSP模擬器 (XC2V250)	對外接頭
Port#0	P0.0~15	P3[16]	JP3[16]
Port#1	P1.0~15	P0[16]	JP2[16]
Port#2	P2.0~15	P2[16]	JP3[16]
ADC	ADC0~7	P2[8]	JP3[8]
FlashROM	D0~7 A14~0 -- WR/RD	XD[8] XA[13] CS/CLK20M XWR/XRD	--
光纖界面	Tx/RxNET Tx/RxUSB	XD[4]	--
系統界面	X1/X2 RESET/ ALARM/	CLK24M XD[2]	-- JP3[2]
其他	--	P1[24] P3[8] XD[2]	JP2[24] JP1[16] JP2[2]
小計	89-pin	122-pin	JP1=16 JP2=42 JP3=42

GPIO界面部分

GPIO界面共有48-bit，和特定界面共用I/O接腳，表列如下：

GPIO	特定界面	GPIO	特定界面	GPIO	特定界面
P0.0	PWM0H (輸出阜)	P1.0	eeSK (輸出阜)	P2.0	(無)
P0.1	PWM1H (輸出阜)	P1.1	eeCS (輸出阜)	P2.1	(無)
P0.2	PWM2H (輸出阜)	P1.2	eeDI (輸出阜)	P2.2	(無)
P0.3	PWM0L (輸出阜)	P1.3	eeDO (輸入阜)	P2.3	(無)
P0.4	PWM1L (輸出阜)	P1.4	ppCK (輸出阜)	P2.4	(無)
P0.5	PWM2L (輸出阜)	P1.5	ppCS (輸出阜)	P2.5	(無)
P0.6	PWMENB(輸出阜)	P1.6	ppDI (輸出阜)	P2.6	(無)
P0.7	PHT1Z (輸入阜)	P1.7	ppDO (輸入阜)	P2.7	(無)
P0.8	PHT1A (輸入阜)	P1.8	ppCLK (輸出阜)	P2.8	(無)
P0.9	PHT1B (輸入阜)	P1.9	ppDIN (輸出阜)	P2.9	(無)
P0.10	PHT0A (輸入阜)	P1.10	RxD (輸入阜)	P2.10	(無)
P0.11	PHT0B (輸入阜)	P1.11	TxD (輸出阜)	P2.11	(無)
P0.12	AMP0(輸入阜)	P1.12	TxEN (輸出阜)	P2.12	(無)
P0.13	AMP1(輸入阜)	P1.13	RxLED (輸出阜)	P2.13	(無)
P0.14	AMP2(輸入阜)	P1.14	TxLED (輸出阜)	P2.14	(無)
P0.15	AMP3(輸入阜)	P1.15	okLED (輸出阜)	P2.15	(無)

其中：

1.P0阜和伺服界面共用I/O接腳，包括PWM輸出、PHT輸入和PWM輸入部分。

2.P1阜和系統界面共用I/O接腳，包括EEPROM、FPGA、RS232和LED界面等等。

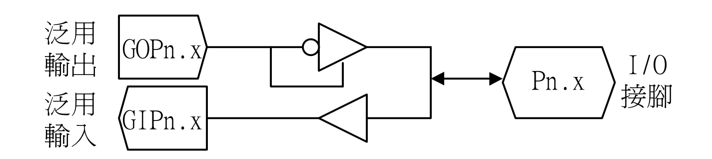
3.P2阜是單獨使用的，無特定界面。

I/O接腳的處理

由於特定界面的不同，GPIO界面的處理分成下述五種，包括：

#1.無特定界面時，這是P2阜的控制模式。
#2.特定界面是無保護的輸出阜時，這是P1阜的輸出控制模式。
#3.特定界面是無保護的輸入阜時，這是P1阜的輸入控制模式。
#4.特定界面是有保護的輸出阜時，這是P0阜的輸出控制模式。
#5.特定界面是有保護的輸入阜時，這是P0阜的輸入控制模式。

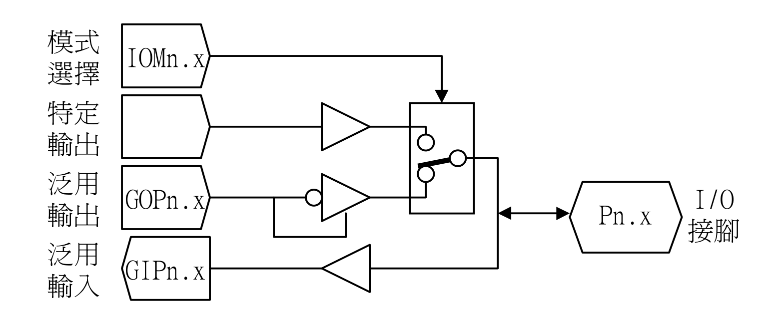
#1：無特定界面時，I/O接腳的處理如下：



其中：

1.泛用輸出點由GOPn.x來控制，採用反向的open-collector結構，
 當GOPn.x=0時，I/O接腳爲open-collector狀態。
 當GOPn.x=1時，I/O接腳輸出爲邏輯0。
2.泛用輸入點由GIPn.x來讀入，但是會因爲GOPn.x的設定而有所不同。
 當GOPn.x=0時，GIPn.x所讀入的就是I/O接腳的輸入狀態。
 當GOPn.x=1時，GIPn.x所讀入的固定爲邏輯0。

#2：特定界面為無保護的輸出阜時，I/O接腳的處理如下：



其中：

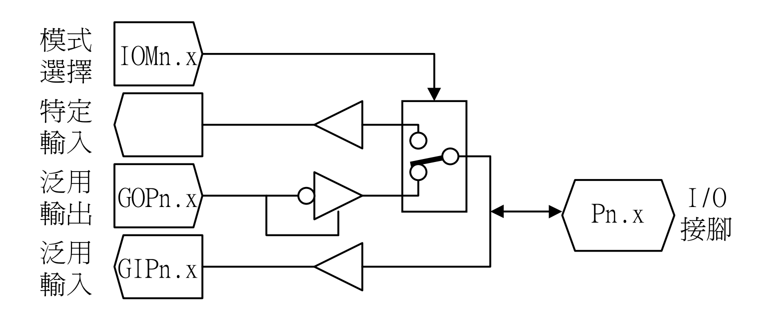
1.控制模式有兩種選擇，

當IOMn.x=0時，選擇泛用I/O模式，使用方式和#1型式相同。

當IOMn.x=1時，選擇特定輸出模式，輸出方式由特定界面來定義。

2.不論IOMn.x的選擇，GIPn.x都可以正確讀出目前的I/O接腳狀態。

#3：特定界面為無保護的輸入阜時，I/O接腳的處理如下：



其中：

1.控制模式有兩種選擇，

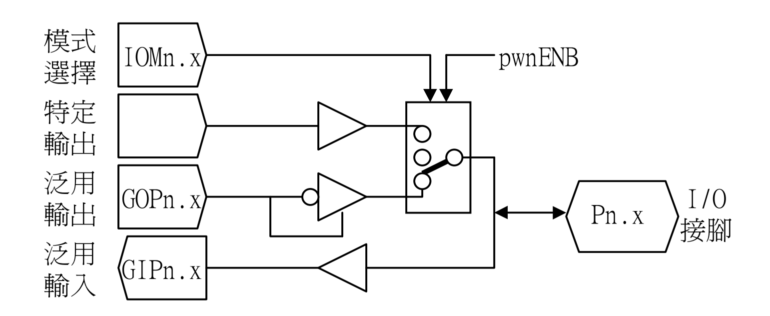
當IOMn.x=0時，選擇泛用I/O模式，使用方式和#1型式相同。

當IOMn.x=1時，I/O接腳直接連到特定輸入，輸入方式由特定界面來定義。

2.不論IOMn.x的選擇，GIPn.x都可以正確讀出目前的I/O接腳狀態。

3.只有當IOMn.x=1時，特定界面才會得到正確輸入，否則特定輸入將固定為0。

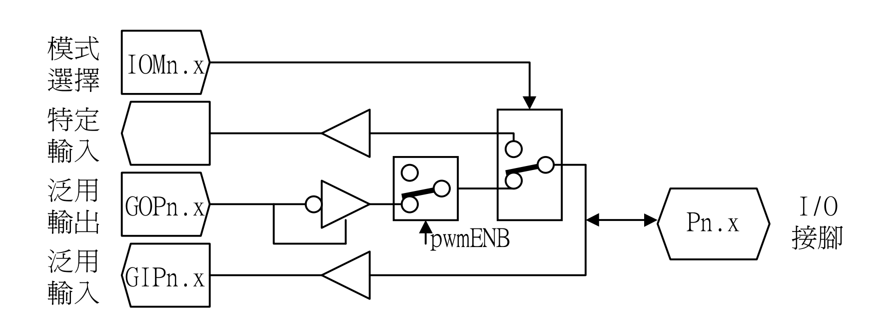
#4：特定界面為有保護的輸出阜時，I/O接腳的處理如下：



其中：

1.控制模式和#2一樣，但是增加了WatchDog的保護，
　當pwmENB=1時，所有輸出都切斷。
　當pwmENB=0時，輸出才能正常工作。
2.不論IOMn.x的選擇，GIPn.x都可以正確讀出目前的I/O接腳狀態。

#5：特定界面為有保護的輸入阜時，I/O接腳的處理如下：



其中：

1.控制模式和#2一樣，但是增加了WatchDog的保護，
　當pwmENB=1時，泛用輸出將被切斷。
　當pwmENB=0時，泛用輸出才能正常工作。
2.不論IOMn.x的選擇，GIPn.x都可以正確讀出目前的I/O接腳狀態。
3.特定輸入界面不受pwmENB的影響。

P0阜的控制

P0阜是16-bit的泛用I/O界面，和PWM輸出控制、PWM輸入解碼和PHT輸入解碼等特定界面共用接腳，表列如下：

GPIO	特定界面	特 定 界 面 功 能 說 明
P0.0	pwm0H (輸出阜)	三相PWM輸出的U相高端IGBT控制
P0.1	pwm1H (輸出阜)	三相PWM輸出的V相高端IGBT控制
P0.2	pwm2H (輸出阜)	三相PWM輸出的W相高端IGBT控制
P0.3	pwm0L (輸出阜)	三相PWM輸出的U相低端IGBT控制
P0.4	pwm1L (輸出阜)	三相PWM輸出的V相低端IGBT控制
P0.5	pwm2L (輸出阜)	三相PWM輸出的W相低端IGBT控制
P0.6	pwmENB(輸出阜)	WatchDog保護狀態的輸出
P0.7	pht0Z (輸入阜)	光電盤#0的Z相輸入點
P0.8	pht0A (輸入阜)	光電盤#0的A相輸入點
P0.9	pht0B (輸入阜)	光電盤#0的B相輸入點
P0.10	pht1A (輸入阜)	光電盤#1的A相輸入點
P0.11	pht1B (輸入阜)	光電盤#1的B相輸入點
P0.12	pwm0IN(輸入阜)	PWM型式電流回授的輸入點
P0.13	pwm1IN(輸入阜)	PWM型式電流回授的輸入點
P0.14	pwm2IN(輸入阜)	PWM型式電流回授的輸入點
P0.15	pwm3IN(輸入阜)	PWM型式電流回授的輸入點

其中：

1. P0阜中的每個bit都可獨立選擇作爲輸入阜、輸出阜或是特定界面。
2. 當IOM0.n=0時，對映的P0.n爲泛用I/O，爲反向open-collector輸出結構。
3. 當IOM0.n=1時，對映的P0.n爲特定界面，其輸入輸出狀況定義如上表。這時若是輸出阜，則是一般的輸出阜，並非open-collect結構。
4. 無論在IOM0/GOP0的任何狀態，GIP0都可得到正確的輸入讀值。
5. 開機啓動設定IOM0=0且GOP0=0，所以在開機時，P0阜固定設爲輸入阜狀態。
6. 當輸出阜爲開集極電路時，通常必須外加10KΩ的pull-up電阻(至3.3V)。
7. 當P0阜設定成輸出阜時會受到WatchDog的影響，只有在pwmENB=0時才准輸出，當pwmENB=1時會切斷所有的輸出界面，必須特別注意。

P1阜的控制

P1阜是16-bit的泛用I/O界面，和EEPROM、FPGA、RS232和LED界面等特定界面共用接腳，表列如下：

GPIO	特定界面	特 定 界 面 功 能 說 明
P1.0	eeSK (輸出阜)	直接連接EEPROM的SK點
P1.1	eeCS (輸出阜)	直接連接EEPROM的CS點
P1.2	eeDI (輸出阜)	直接連接EEPROM的DI點
P1.3	eeDO (輸入阜)	直接連接EEPROM的DO點
P1.4	ssCK (輸出阜)	直接連接FPGA串列界面的CK點
P1.5	ssCS (輸出阜)	直接連接FPGA串列界面的CS點
P1.6	ssDI (輸出阜)	直接連接FPGA串列界面的DI點
P1.7	ssDO (輸入阜)	直接連接FPGA串列界面的DO點
P1.8	ppCLK (輸出阜)	直接連接FPGA下載界面的CCLK點
P1.9	ppDIN (輸出阜)	直接連接FPGA下載界面的DIN點
P1.10	RxD (輸入阜)	RS232界面的RxD輸入
P1.11	TxD (輸出阜)	RS232界面的TxD輸出
P1.12	TxDEN (輸出阜)	RS232界面的TxD輸出控制
P1.13	RxPLED(輸出阜)	光纖輸入端(RxP)的LED顯示
P1.14	TxPLED(輸出阜)	光纖輸出端(TxP)的LED顯示
P1.15	stsLED(輸出阜)	DSP控制狀態的LED顯示

其中：

1. P1阜中的每個bit都可獨立選擇作爲輸入阜、輸出阜或是特定界面。
2. 當IOM1.n=0時，對映的P1.n爲泛用I/O，爲反向open-collector輸出結構。
3. 當IOM1.n=1時，對映的P1.n爲特定界面，其輸入輸出狀況定義如上表。這時若是輸出阜，則是一般的輸出阜，並非open-collect結構。
4. 無論在IOM1/GOP1的任何狀態，GIP1都可得到正確的輸入讀值。
5. 開機啓動設定IOM1=0且GOP1=0，所以在開機時，P1阜固定設爲輸入阜狀態。
6. 當輸出阜爲開集極電路時，通常必須外加10KΩ的pull-up電阻(至3.3V)。
7. P1阜不會受到WatchDog的影響，無論pwmENB爲何都會有正確輸出。

P2阜的控制

P2阜是16-bit的泛用I/O界面，並無其他特定用途，其中：

1.P2阜中的每個bit都可獨立選擇作爲輸入阜或是輸出阜。

2.當設定爲輸出阜時，P2阜爲open-collector的輸出結構，即是GOP2參數中對映bit的反向輸出結果。所以

　當GOP2.n=0時，其對映的P2.n爲浮動狀態，這時P2.n可作爲輸入阜使用。

　當GOP2.n=1時，其對映的P2.n爲邏輯0輸出，即是反向的輸出阜功能。

4.無論GOP2的任何狀態，GIP2都可得到正確的輸入讀值，其中

　當GOP2.n=0時，其對映的GIP2.n所讀入的是P2.n的外部輸入訊號。

　當GOP2.n=1時，其對映的GIP2.n所讀入的是GOP2.n的反向輸出訊號。

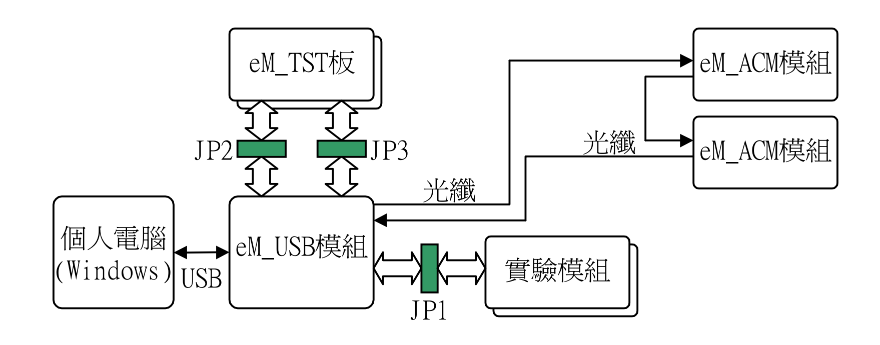
5.開機啓動設定GOP2=0，所以在開機時，P2阜固定設爲輸入阜狀態。

6.由於輸出阜爲開集極電路，通常必須外加10KΩ的pull-up電阻(至3.3V)。

7.當P2阜設定成輸出阜時，並不會受到WatchDog的影響。

電路板配置

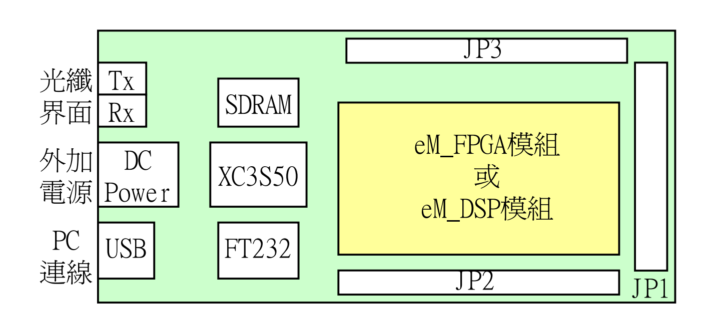
eM_USB模組外部連線的系統方塊圖如下：



其中：

1.eM_USB模組以USB界面和個人電腦連線。
2.eM_USB模組的對外界面包括：
  A.以26P接頭(JP1)連接既有的實驗模組。
  B.以50P接頭(JP2/3)連接上層的測試板(eM_TST)，提供界面的測試功能。
  C.以光纖接頭連接各個伺服驅動模組(eM_ACM)。

而eM_USB模組本身的電路配置圖如下：



其中：

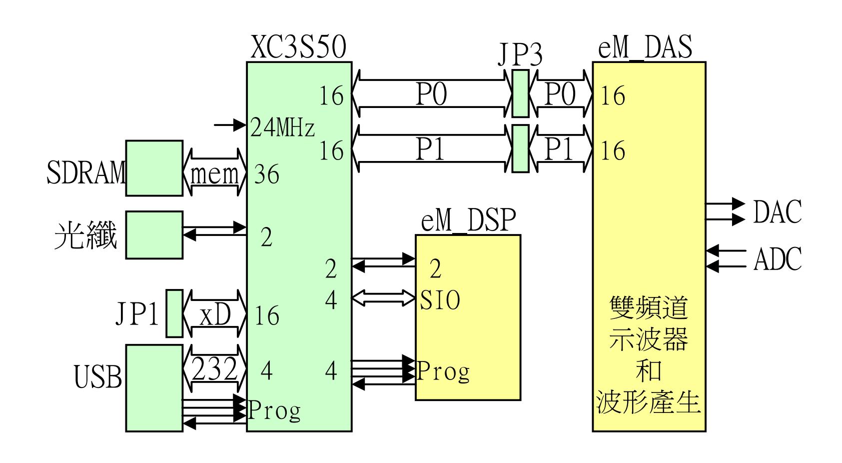
1.FT232BM提供USB對RS232的界面轉換工作，而SDRAM提供擴張的記憶體空間，可儲存大量的循序命令或是測試記錄。
2.XC3S50負責eM_USB模組中所有的界面處理，可以針對不同應用作彈性的設計。
3.可用插卡型式擴接真正的控制模組，可以選擇eM_FPGA模組作FPGA實驗應用，或是eM_DSP模組作DSP實驗應用。

連線規劃

應用上分成兩類目標在規劃，包括：

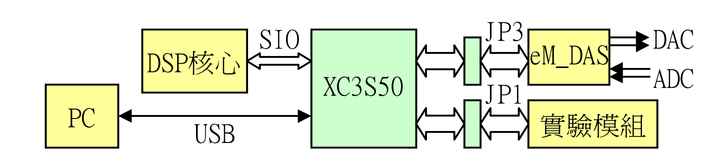
1.針對既有教學實驗，目標為取代現有的eMotion模組。
2.針對CNC控制系統，目標在運動控制模組和DSP元件測試。

針對既有教學實驗作規劃的連線關係如下：



其中：

1.以XC3S50為控制主體，當需要DSP核心時可再插上eM_FPGA卡或是eM_DSP卡。
2.主要應用是透過JP1界面來連接既有的實驗模組，並透過上層的eM_DAS卡提供雙頻道的示波器和波形產生器功能。方塊圖如下：

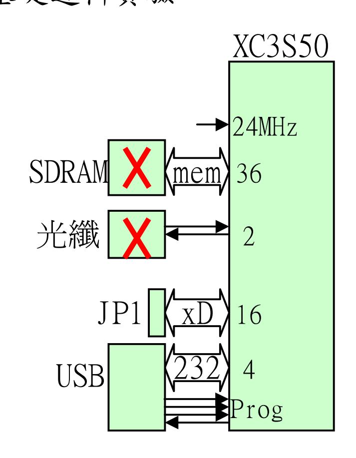


3.eM_FPGA只是提供DSP核心的程式運算功能，所有I/O界面都放棄使用權，並透過光纖界面和XS3S50及PC作連線。
4.eM_DAS板提供高速AD/DA界面，抽樣速度可高達24MHz，以提供各項實驗中類比電壓的測試功能。

這方面的應用範例包括：

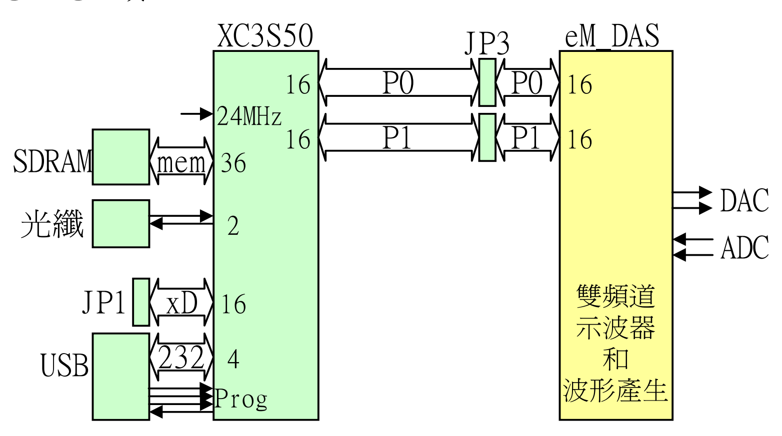
1.基礎邏輯實驗。
2.數位通訊實驗。
3.電力電子實驗。
4.單純的示波器和波形產生器。

範例A：基礎邏輯實驗



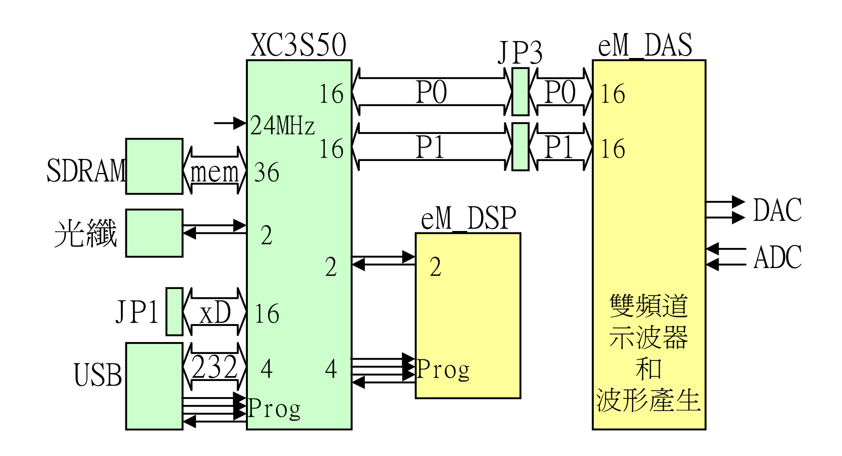
1.只有eM_USB空板，其他模組都不必安裝。
2.控制目標爲XC3S50，可透過Xilinx新版的免費工具進行開發。
3.可透過JP1連接其他既有的實驗模組。
4.SDRAM和光纖界面雖已安裝在電路板中，但在實驗中均不需理會。
5.一般而言，USB界面只負責FPGA下載的工作，只有在應用實驗中需要再增加RS232界面解碼以執行實驗中的參數設定工作。

範例B：電力電子實驗



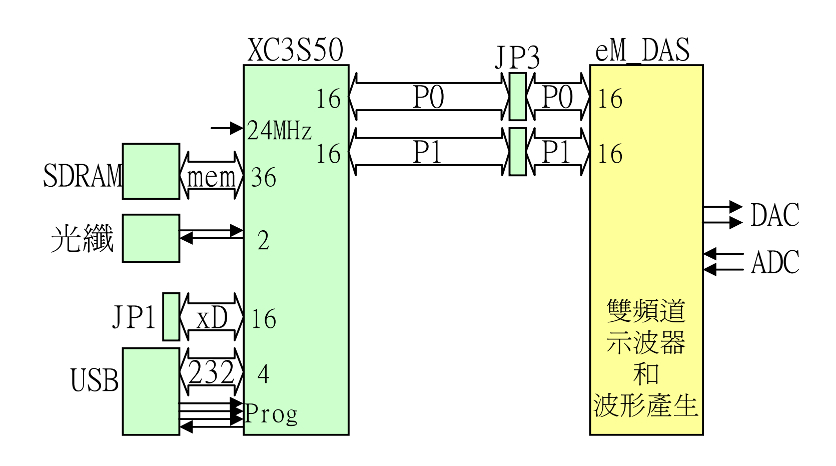
1.安裝eM_DAS模組，提供雙頻道的ADC作量測，並提供雙頻道的DAC作電壓設定。
2.ADC的抽樣頻率高達24MHz，可量測實驗中的所有波形。
3.實驗中以XC3S50爲設計主體，可透過Xilinx新版的免費工具進行開發。
4.透過JP1連接實驗模組(eM_DC2DC模組)。
5.USB界面負責電路下載和參數設定工作，並執行示波器和邏輯分析儀的功能。

範例C：數位通訊實驗



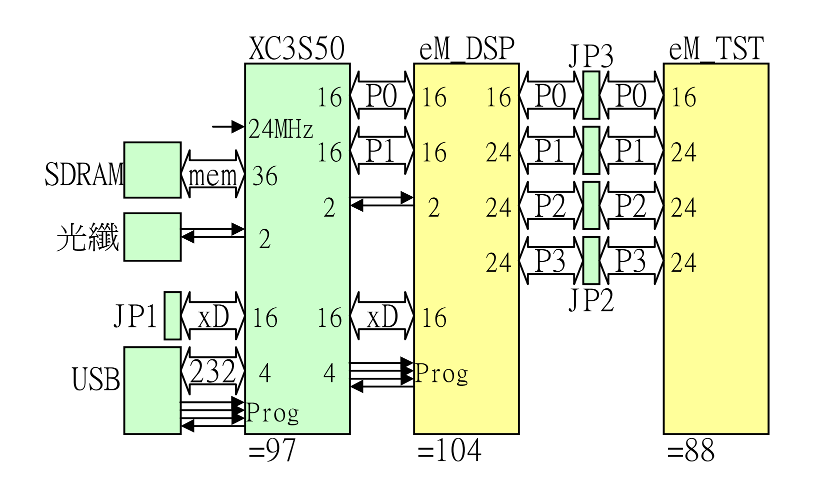
1.安裝eM_DAS模組，提供雙頻道的ADC作量測，並提供雙頻道的DAC作電壓設定。
2.ADC的抽樣頻率高達24MHz，可量測實驗中的所有波形。
3.實驗中以XC3S50爲設計主體，可透過Xilinx新版的免費工具進行開發。
4.透過JP1連接實驗模組(eM_EXP1和eM_EXP2模組)。
5.需要DSP核心作運算的實驗，可再插上eM_DSP板作執行。
6.USB界面負責電路下載和參數設定工作，並執行示波器和邏輯分析儀的功能。

範例D：雙頻道的示波器和波形產生器



1.安裝eM_DAS模組，提供雙頻道的ADC作示波器量測，並提供雙頻道的DAC作波形產生器。
2.ADC和DAC的抽樣頻率高達24MHz，可量測和控制實驗中的所有波形。
3.所有軟硬體工作都已事先完成，使用者可以直接在Matlab中使用。

針對CNC控制作規劃的連線關係如下：



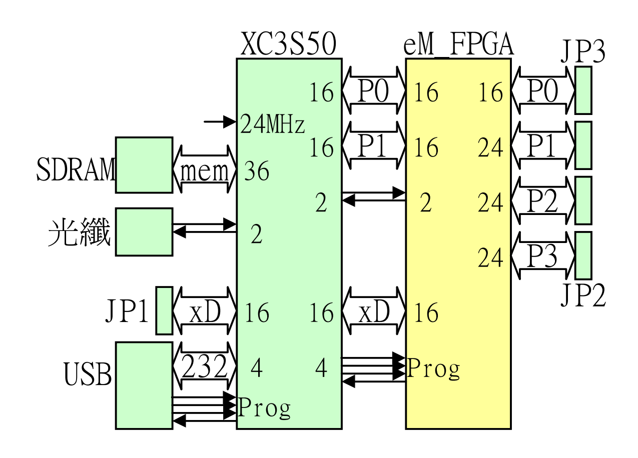
其中：

1.以eM_DSP爲控制主體，可插上eM_FPGA卡作DSP模擬器或是直接插上eM_DSP卡作晶片測試。
2.DSP中所有的I/O界面可透過JP3接頭全部接出，並可插上eM_TST板作LED/KEY的簡單顯示和控制，提供初學者做程式練習。
3.XS3S50提供USB界面和邏輯分析儀的功能，可提供48M-word/sec的記錄速度。
4.可透過光纖網路測試其他的伺服驅動模組。

這方面的應用範例包括：

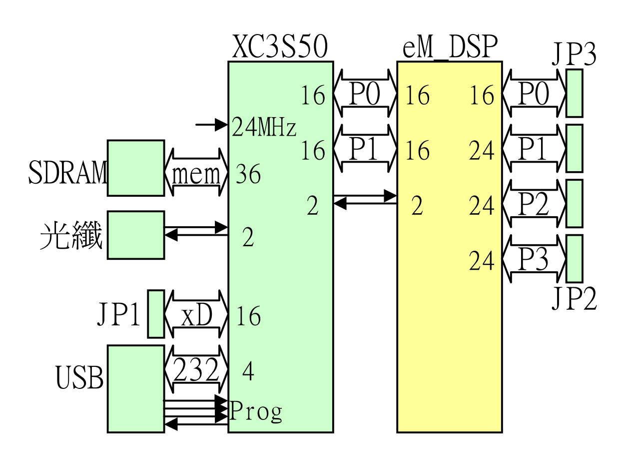
1.DSP模擬器的開發與測試。
2.DSP晶片的測試。
3.DSP晶片的教學實驗。
4.基本的運動控制模組。
5.擴張的運動控制模組。
6.基礎邏輯實驗(FPGA和CPLD)。
7.進階邏輯實驗(DSP核心設計)。
8.直流馬達控制實驗。
9.交流馬達控制實驗。

範例A：DSP模擬器的開發與測試



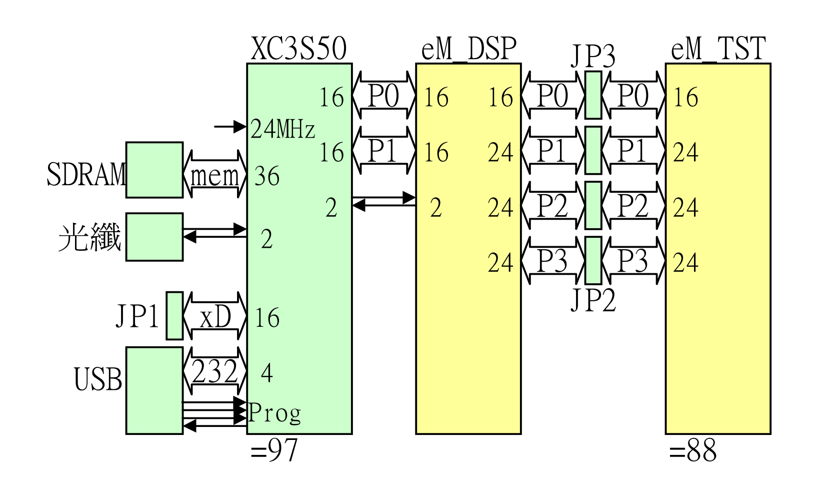
1.插上eM_FPGA卡，固定選擇XC2V250。
2.XC2V250可透過Xilinx新版的免費工具進行開發。
3.所有I/O都可透過JP2/JP3進行量測，還可透過XC3S50間的50-pin界面進行測試工作，包括邏輯分析儀的測試和記錄。
4.必要時可在JP2/JP3上插上其他的測試卡進行測試。

範例B：DSP晶片的測試



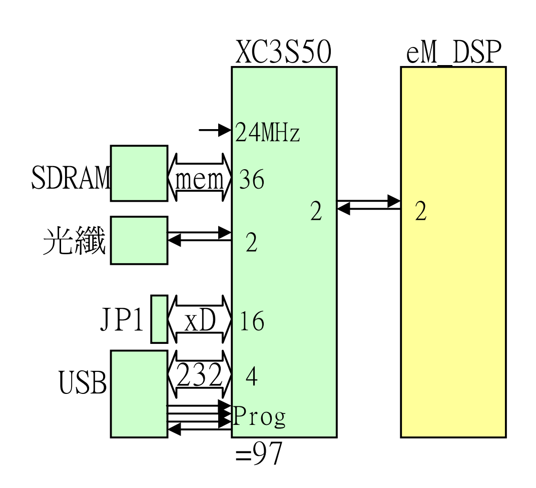
1.換插上eM_DSP卡，即可測試新開發的DSP晶片。
2.所有I/O都可透過JP2/JP3進行量測，還可透過XC3S50間的34-pin界面進行測試工作，包括邏輯分析儀的測試和記錄。
3.程式和資料的下載都可透過USB界面處理。
4.必要時可在JP2/JP3上插上其他的測試卡進行測試。

範例C：DSP晶片的教學實驗



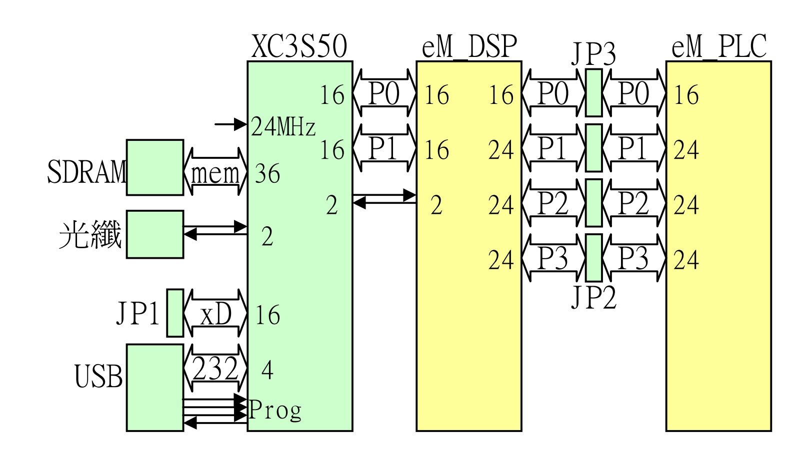
1.插上eM_DSP卡，即可測試新開發的DSP晶片。
2.所有I/O都可透過JP2/JP3進行量測。
3.eM_TST卡上包括簡單的LED/KEY/RS232，以及EEPROM和FlashROM，可透過程式撰寫來進行測試和實習。
4.XC3S50提供USB界面，所有DSP晶片中的程式和資料都可透過USB界面作下載。另外還可透過其中的34-pin界面進行邏輯分析儀的測試和記錄工作。

範例D：基本的運動控制模組(DSP開發工具)



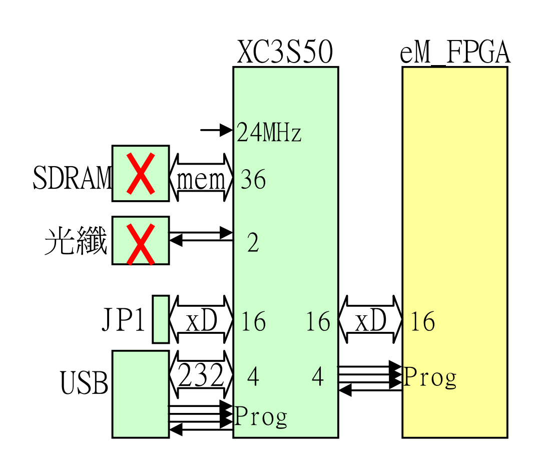
1.基本的運動控制模組就是將來的DSP開發工具，可透過光纖網路連接其他的伺服驅動模組。
2.主要I/O就是光纖網路，並由XC3S50提供USB界面，所有DSP晶片中的程式和資料都可透過USB界面和光纖網路作下載及監控。
3.eM_DSP卡中的DSP晶片提供輔節點#1的功能，可作為運動控制中的循軌控制工作，並協調所有其他的伺服驅動模組。

範例E：擴張的運動控制模組



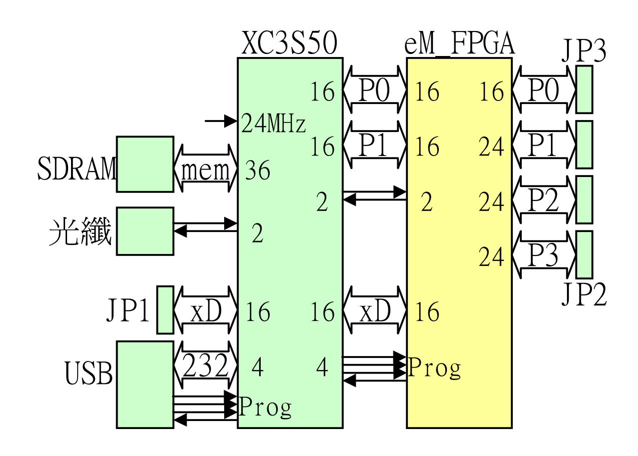
1.擴張的運動控制模組就是將基本運動控制模組再加上PLC控制的功能。
2.藉由JP2/JP3將DSP晶片中的所有I/O引出，連接eM_PLC板來執行PLC控制功能。
3.由於USB接線長度(<2m)的限制，這種PLC控制器必須安置在PC旁邊，所以主要的應用市場就是CNC控制器中的操作面板控制，包括按鍵、LED和手輪。

範例F：基礎邏輯實驗(其他FPGA或CPLD)



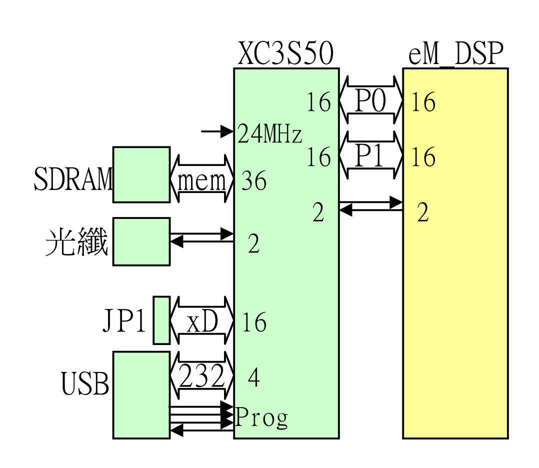
1.插上eM_FPGA卡，可換用不同的FPGA或CPLD作實驗。
2.控制目標爲eM_FPGA中的FPGA或CPLD，透過Xilinx新版的免費工具進行開發。
3.可透過JP1連接其他既有的實驗模組。
4.SDRAM和光纖界面雖已安裝在電路板中，但在實驗中均不需理會。
5.一般而言，USB界面只負責FPGA下載的工作，只有在應用實驗中需要再增加RS232界面解碼以執行實驗中的參數設定工作。

範例G：進階邏輯實驗(DSP核心設計)



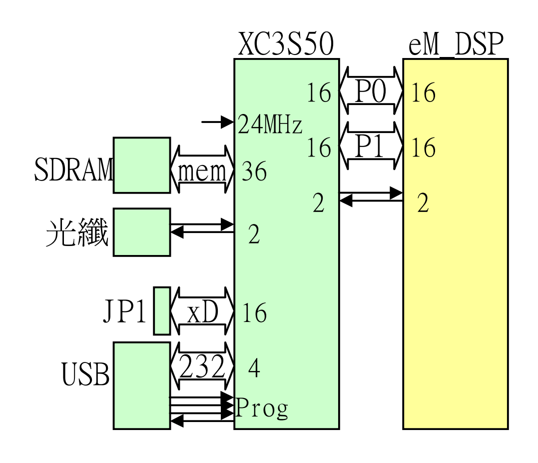
1.插上eM_FPGA卡，固定選用XC2V250作實驗。
2.控制目標爲XC2V250，透過Xilinx新版的免費工具進行開發。
3.可透過JP1連接其他既有的實驗模組，或從JP2/JP3上進行量測。
4.XC3S50提供USB界面，所有DSP晶片中的程式和資料都可透過USB界面作下載。另外還可透過其中的50-pin界面進行邏輯分析儀的測試和記錄工作。

範例H：直流馬達控制的教學實驗



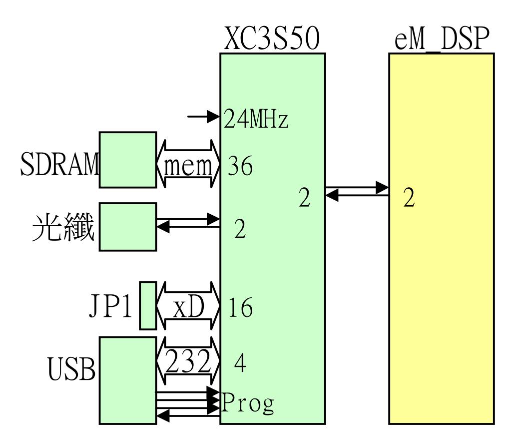
1.以DSP晶片進行直流馬達的程式控制工作。
2.直流馬達界面是透過XC3S50再從JP1接頭送出，連接既有的直流馬達模組(eM_DCM模組)。
3.XC3S50提供USB界面，所有DSP晶片中的程式和資料都可透過USB界面作下載。另外還可透過其中的34-pin界面進行示波器和邏輯分析儀的測試工作。

範例H：交流馬達控制的教學實驗(低壓實驗)



1. 以DSP晶片進行直流馬達的程式控制工作。
2. 交流馬達界面是透過XC3S50再從JP1接頭送出，連接既有的交流馬達模組(eM_DC2AC模組)。
3. XC3S50提供USB界面，所有DSP晶片中的程式和資料都可透過USB界面作下載。另外還可透過其中的34-pin界面進行示波器和邏輯分析儀的測試工作。
4. 由於eM_DC2AC模組只能接受60VDC電源，算是比較安全的實驗環境，適合初學者作實驗。

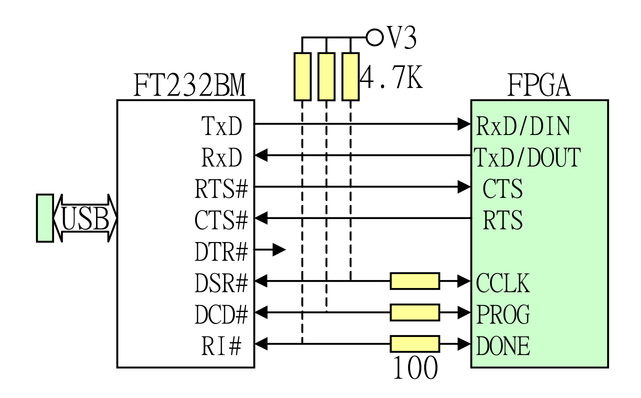
範例H：交流馬達控制的教學實驗(高壓實驗)



1. 當成DSP發展工具來使用，真正的實驗目標是透過光纖網路再連接的功率驅動模組(eM_ACM模組)。
2. XC3S50提供USB界面，所有DSP程式和資料都可透過USB界面和光纖網路下載，還可執行示波器和邏輯分析儀的測試工作。
3. eM_ACM模組直接在110VAC的環境下工作，比較適合研究所和專題製作的學生。

USB界面設計

在USB界面部分是利用FT232BM作界面連線，圖示如下：



1.FT232BM負責作USB界面對RS232界面的轉換。在RS232端的訊號包括：

TxD (out)：Transmit Data Output
RxD (in) ：Receive Data Input
RTS#(out)：Request To Send (Handshake Signal)
CTS#(in) ：Clear To Send (Handshake Signal)
DTR#(out)：Data Terminal Ready (Handshake Signal)
DSR#(in) ：Data Set Ready (Handshake Signal)
DCD#(in) ：Data Carrier Detect
RI# (in) ：Ring Indicator

2.在正常工作時，RS232端的通訊規格爲(3M/8/N/2)，其中：

Baudrate：3Mbit/sec
Length ：8-bit
Stop Bit：2-bit
Parity ：Even Parity

而流量控制設定爲(FT_FLOW_RTS_CTS)，即是以RTS/CTS爲流量控制點。

3.所以在正常狀態時，USB界面是透過(TxD/RxD/RTS#/CTS#)四點來執行RS232的串列通訊工作，其他(DTR#/DSR#/DCD#/RI#)四點將不會影響通訊結果。

4.在FPGA下載時，FT232BM將進入BitMode模式，這時RS232端的八點訊號將視爲單純的並列訊號作處理。這時設定

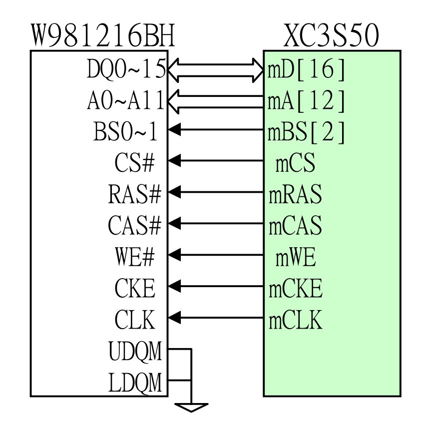
TxD(bit0=in)，RxD(bit1=in)，RTS#(bit2=in)，CTS#(bit3in)
DTR#(bit4=out)，DSR#(bit5=out)，DCD#(bit6=out)，RI#(bit7=in)

而由DTR#/DSR#/DCD#/RI#)四點來執行FPGA的下載工作。

5.由於FPGA的下載控制(DIN/CCLK/PROG/DONE)訊號爲2.5V邏輯，所以必須再外加100Ω串聯電阻作訊號保護用。

SDRAM界面設計

SDRAM採用華邦的W981216BH，和FPGA之間的連線圖示如下：



1.W981216BH是128M-bit的動態RAM，內部分割為
    2M-word * 4-bank * 16-bit
  其中bank由BS0~1來選擇，而位址由Row(12-bit)和Column(9-bit)來選擇。
2.匯流排固定為16-bit的格式，所以將UDQM和LDQM都固定為0，維持在永遠ON的狀態。換句話說，SDRAM可視為8M-word的記憶空間來使用。
4.為了加快讀寫速度，每次讀寫都以4-word為單位，目標是將寫入速度提高到48M-word/sec的速度範圍。如果當成雙頻道示波器來使用時，其記錄速度可達24MHz。如果當成16-bit的邏輯分析儀來使用時，其記錄速度也可達48MHz。
5.如果以48MHz的速度記錄16-bit資料，可記錄160ms的動態資料。
6.在執行CNC控制工作時，SDRAM可當成DNC命令的緩衝區，最多可儲存8M-word的DNC命令。

SDRAM的控制程序有多種選擇，非常複雜。但是我們需求是固定的，所以只要使用其中的部分功能即可，依序說明如下。

在讀取時，採用8-byte的Burst-Mode作處理，表列如下：

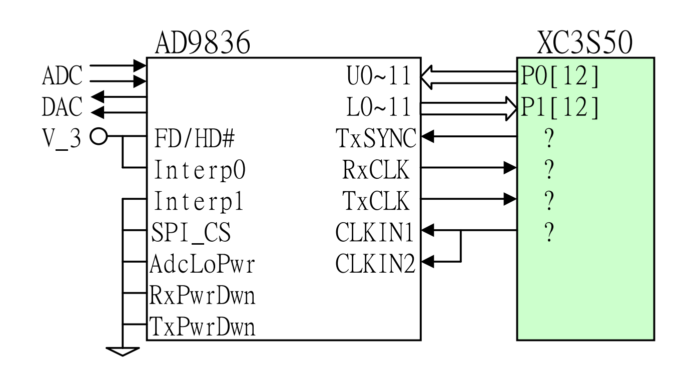
	CKE	DQM	BS0,1	A10	A0~11	CS#	RAS#	CAS#	WE#	DQ[8]
#1	1	sel	sel	RA	RA	0	0	1	1	-
#2	1	sel	sel	RA	RA	1	1	1	1	-
#3	1	sel	sel	1	CA	0	1	0	1	-
#4	1	sel	sel	1	CA	1	1	1	1	-
#5	1	sel	x	x	x	1	1	1	1	D0
#6	1	sel	x	x	x	1	1	1	1	D1
#7	1	sel	x	x	x	1	1	1	1	D2
#8	1	sel	x	x	x	1	1	1	1	D3
#9	1	sel	x	x	x	1	1	1	1	D4
#10	1	sel	x	x	x	1	1	1	1	D5
#11	1	sel	x	x	x	1	1	1	1	D6
#12	1	sel	x	x	x	1	1	1	1	D7

其中：

1.採用Burst=8、Mode=Interleave、Laterncy=2的讀寫模式。
2.讀取程序依序爲：
   A.設定RA的位址。
   B.等待一個週期(因爲Laterncy=2)。
   C.設定CA的位址。
   D.等待一個週期(因爲Laterncy=2)。
   E.開始依序讀入8-byte的資料。

高速ADDA界面設計

高速ADDA採用Analog Device的AD9863，和FPGA之間的連線圖示如下：



1. AD9863是64-pin的晶片，提供雙頻道12-bit的ADC界面，以及雙頻道12-bit的DAC界面。其ADC的抽樣頻率高達50MHz，而DAC的抽樣頻率更高達200MHz。
2. 我們藉由AD9863提供雙頻道的示波器功能，目標是能提供雙頻道24MHz的抽樣和記錄速度。
3. AD9863並非裝在eM_USB模組上，而是裝在以JP3連接的擴張模組(eM_DAS)中。
4. 和W981216BH是128M-bit的動態RAM，內部分割為

   2M-word * 4-bank * 16-bit

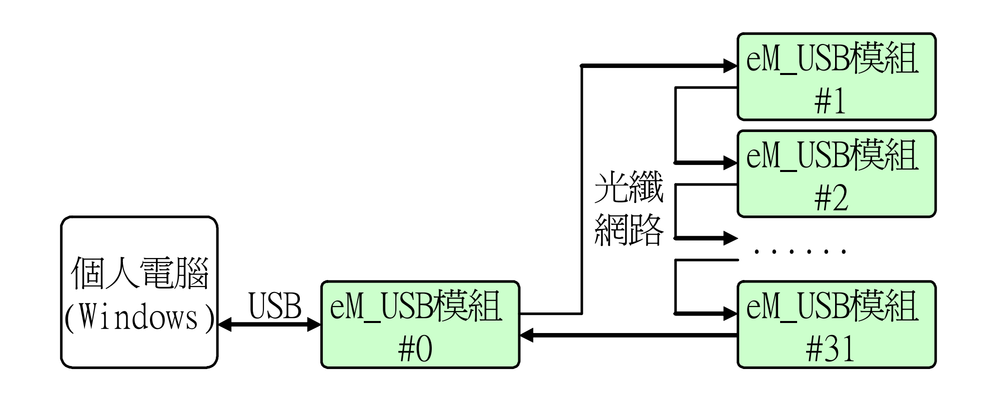
   其中bank由BS0~1來選擇，而位址由Row(12-bit)和Column(9-bit)來選擇。

附錄二. 光纖網路式FPGA發展系統

唐佩忠

系統配置

光纖網路式FPGA發展系統的系統配置圖如下：

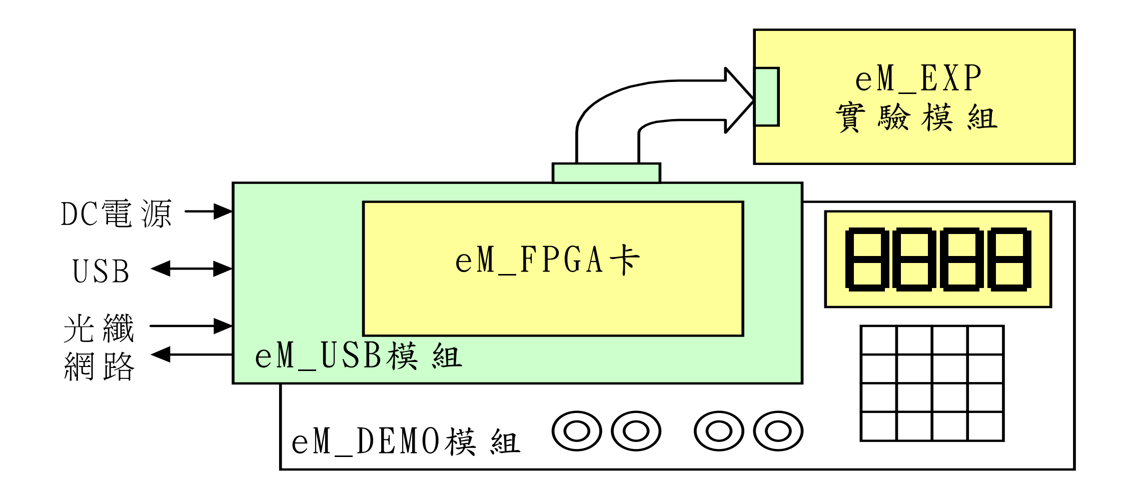


其中：

1. 基本模組爲eM_USB模組，可用USB界面和個人電腦連線，並提供USB對光纖網路的轉換功能，使得個人電腦可以獨立控制光纖網路中的每個網路節點。
2. eM_USB模組也可以作爲光纖網路中的網路節點，透過光纖網路而接受個人電腦的監控。
3. 就個人電腦而言，不論是USB界面直接連結或是以光纖網路間接連結的eM_USB模組，都是可以直接監控的網路節點。最多可以同時控制一個USB節點和30個網路節點。
4. 對於每一個網路節點，個人電腦都可執行：
   eM_USB模組中FPGA的電路下載工作，並在適當的FPGA電路配合下，可執行eM_USB模組中FPGA內部電路的讀寫控制。
5. 透過適當的FPGA電路設計，各網路節點都可透過光纖網路和其他任何網路節點作同步通訊，執行FPGA內部記憶體的資料交換。資料交換頻率可達1KHz以上。

網路模組配置

每個網路模組的標準配置圖如下：



其中：

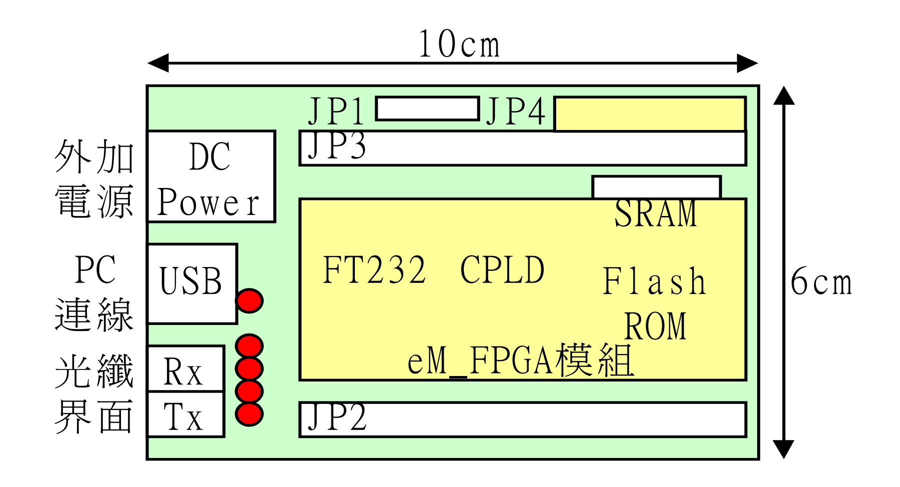
1. 網路模組以eM_USB模組為主，提供基本的USB界面和光纖網路連線。
2. eM_USB模組中的FPGA/CPLD採用插卡型式，可選擇不同的eM_FPGA卡，包括：
   eM_2V1K卡：可選用Virtex系列的XC2V250和XC2V1000。
   eM_3S1K卡：可選用Spartan3系列的XC3S400、XC3S1000和XC3S1500。
   eM_3S50卡：可選用Spartan3系列的XC3S50、XC3S200和XC3S400。
   eM_95XL卡：可選用9500XL系列的XC95144XL和XC95288XL。
3. eM_USB模組中的對外連線有兩種。一種是26P接頭，用以連接既有的各種實驗模組。另一種是兩組50P接頭，將FPGA中各接腳引出，用以連接專題實驗。
4. eM_DEMO模組就是插上50P接頭的教學模組，其中包括LED數字顯示和4*4鍵盤。也具備兩組的ADC和兩組的DAC接頭，提供類比訊號的測試工具。

常用的實驗組合包括：

1. 精簡組合：包括(eM_USB+eM_FPGA)，提供基本的網路節點，適用於專題實驗。
2. 標準組合：包括(eM_USB+eM_FPGA+eM_DEMO)，提供基礎FPGA設計的教學實驗。
3. 實驗組合：包括(eM_USB+eM_FPGA+eM_DEMO+eM_EXP)，提供進階的FPGA實驗。
   可依需求選用不同的實驗模組。

eM_USB模組

eM_USB模組的電路配置圖如下：



其中：

1.對外的連線接頭包括：

DC電源　：可選擇單電源(5V)或是多電源(5V/+12V/-12V)兩種

USB界面　：直接連接個人電腦

光纖界面：連接光纖網路

2.實驗界面包括：

JP1：負責內部CPLD的電路下載

JP2/JP3：兩組50P的擴張用接頭

JP4　　：標準26P接頭，用以連接各種實驗模組。

3.內部的電路配置包括：

A.以SO-DIMM接頭連接各種eM_FPGA卡。

B.內附EEPROM、FlashROM和SRAM，提供FPGA作應用測試。

C.以一顆FT232BM作爲USB界面解碼，而以一顆CPLD作爲光纖網路解碼。

D.提供各種eM_FPGA卡的電路下載功能。

4.內附五顆LED，由上而下依序爲：

usbLED：指示USB界面的讀寫狀態

tstLED：可由FPGA電路自行定義

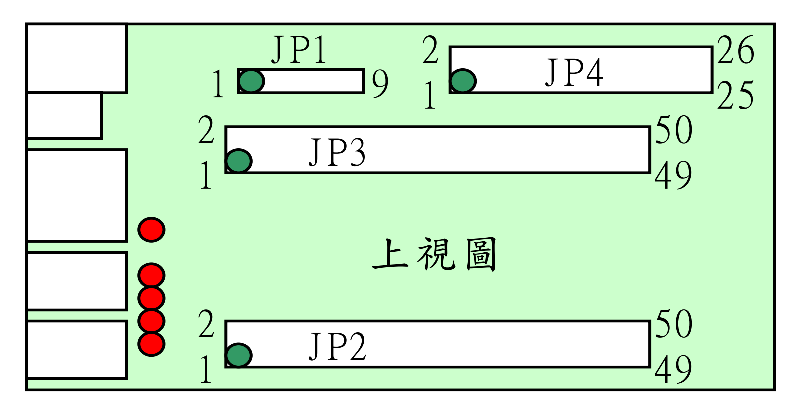
stsLED：可由FPGA電路自行定義

RxLED　：可由FPGA電路自行定義

TxLED　：可由FPGA電路自行定義

外部接腳定義

外部接頭的接點順序圖示如下：

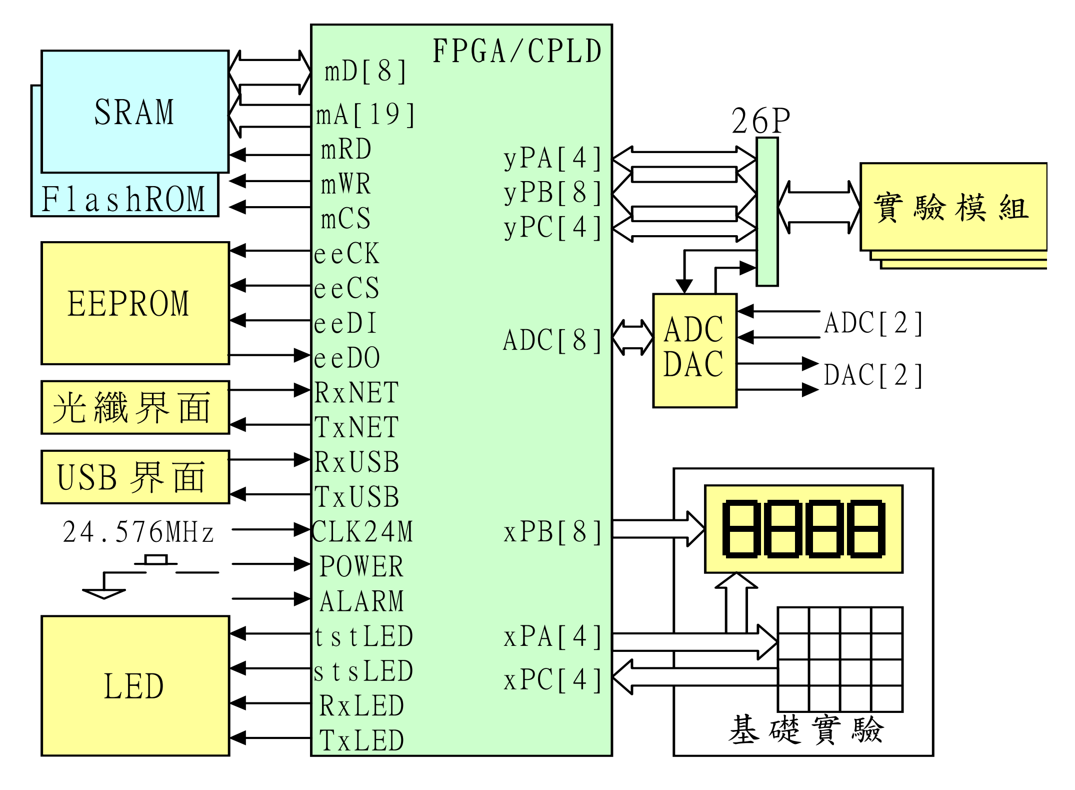


而各接頭的接點定義如下：

JP1	JP4		JP2		JP3	
1:+3.3V	1:+12V	2:+12V	1: GND	2: GND	1: GND	2: GND
2:GND	3: ADC	4: DAC	3:POWER	4:ALARM	3: +5V	4: +5V
3:nc	5: yPB2	6: yPB3	5:RxNET	6:TxNET	5:+12V	6:+12V
4:TCK	7: yPB1	6: yPB4	7:RxUSB	6:TxUSB	7:-12V	6:-12V
5:nc	9: yPB0	10:yPB5	9:RxCAN	10:TxCAN	9: ADC	10: DAC
6:TDO	11:yPB6	12:yPB7	11:P0D0	12:P0D1	11:P1D0	12:P1D1
7:TDI	13:yGND	14:yPC3	13:P0D2	14:P0D3	13:P1D2	14:P1D3
8:nc	15:yPC2	16:yPC1	15:P0D4	16:P0D5	15:P1D4	16:P1D5
9:TMS	17:yPC0	18:yPA0	17:P0D6	18:P0D7	17:P1D6	18:P1D7
	19:yPA1	20:yPA2	19:P0D8	20:P0D9	19:P1D8	20:P1D9
	21:yPA3	22:GND	21:P0D10	22:P0D11	21:P1D10	22:P1D11
	23:GND	24:+5V	23:P0D12	24:P0D13	23:P1D12	24:P1D13
	25:+5V	26:nc	25:P0D14	26:P0D15	25:P1D14	26:P1D15
			27:P2D0	28:P2D1	27:PxA0	28:PxA1
			29:P2D2	30:P2D3	29:PxA2	30:PxA3
			31:P2D4	32:P2D5	31:PxA4	32:PxA5
			33:P2D6	34:P2D7	33:PxA6	34:PxA7
			35:P2D8	36:P2D9	35:PxB0	36:PxB1
			37:P2D10	38:P2D11	37:PxB2	38:PxB3
			39:P2D12	40:P2D13	39:PxB4	40:PxB5
			41:P2D14	42:P2D15	41:PxB6	42:PxB7
			43:ADC0	44:ADC1	43:PxC0	44:PxC1
			45:ADC2	46:ADC3	45:PxC2	46:PxC3
			47:ADC4	48:ADC5	47:PxC4	48:PxC5
			49:DAC0	50:DAC1	49:PxC6	50:PxC7

eM_FPGA模組

在各種eM_FPGA模組中，FPGA對外電路的標準配置圖如下：



其中：

1.FPGA/CPLD可以選擇下述幾種：

XC2V250/XC2V1000　　　　　　：採用eM_2V1K模組，其中FPGA爲FG456包裝。
XC3S400/XC3S1000/XC3S1500：採用eM_3S1K模組，其中FPGA爲FG456包裝。
XC3S400/XC3S1000/XC3S1500：採用eM_3S1K模組，其中FPGA爲FG456包裝。
XC3S50/XC3S200/XC3S400　　　：採用eM_3S50模組，其中FPGA爲TQ144包裝。
XC95144XL/XC95288XL　　　　：採用eM_95XL模組，其中CPLD爲TQ144包裝。

2.採用FG456包裝者可以控制所有的I/O界面。

3.採用TQ144包裝者只是不能控制RAM和FlashROM，而可以控制其他所有的界面。

4.所有FPGA模組可自由選擇由USB界面或是由光纖界面下載電路。但是CPLD模組只能固定由USB界面下載電路。

5.所有FPGA/CPLD模組都可透過26P接頭連接既有的各種實驗模組。

FPGA接腳對照表

eM_DEMO 電路配置	eMotion 舊版定義	piDSP 新版定義	XC2V250 -FG456	XC3S1000 -FG456	XC3S200 -TQ144	XC95144XL -TQ144
mD0	XD6	mD0	A17	C13	--	--
mD1	XD7	mD1	B17	D12	--	--
mD2	XD8	mD2	C17	D11	--	--
mD3	XD9	mD3	D17	C11	--	--
mD4	XD10	mD4	E17	D10	--	--
mD5	XD11	mD5	E16	C10	--	--
mD6	XD12	mD6	E14	D9	--	--
mD7	XD13	mD7	E13	D7	--	--
mA0	XD14	mA0	D13	C7	--	--
mA1	XD15	mA1	C13	D6	--	--
mA2	XA0	mA2	V5	C3	--	--
mA3	XA1	mA3	U5	C4	--	--
mA4	XA2	mA4	Y2	D3	--	--
mA5	XA3	mA5	Y1	D4	--	--
mA6	XA4	mA6	V4	E3	--	--
mA7	XA5	mA7	V3	E4	--	--
mA8	XA6	mA8	W2	F3	--	--
mA9	XA7	mA9	W1	F4	--	--
mA10	XA8	mA10	T2	G6	--	--
mA11	XA9	mA11	T1	K3	--	--
mA12	XA10	mA12	R4	K10	--	--
mA13	XA11	mA13	R3	L3	--	--
mA14	XA12	mA14	R2	L4	--	--
mRD	XRD#	mRD	R1	D5	--	--
mWR	XWR#	mWR	P6	C6	--	--
POWER	P0A0	POWER	B4	B20	P1	P5
ALARM	P0A1	ALARM	A4	A19	P2	P6
RxNET	P0A2	RxNET	C4	B19	P4	P7
TxNET	P0A3	TxNET	C5	A18	P5	P9
RxUSB	P0B0	RxUSB	B5	A15	P6	P10
TxUSB	P0B1	TxUSB	A5	B15	P7	P11
RxCAN	P0B2	RxCAN	D6	A14	P8	P12
RxCAN	P0B3	TxCAN	C6	B14	P10	P13
CLK24M	CLK24M	CLK24M	E12	B12	P124	P38

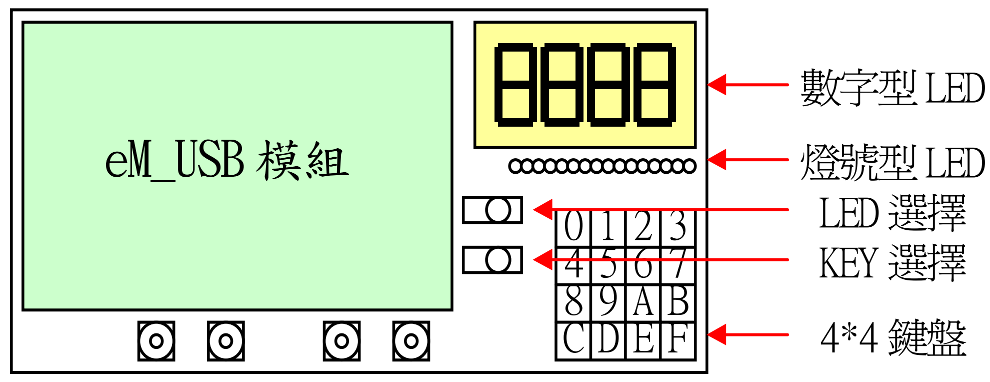
eM_DEMO 電路配置	eMotion 舊版定義	piDSP 新版定義	XC2V250 -FG456	XC3S1000 -FG456	XC3S200 -TQ144	XC95144XL -TQ144
PWM0H	P3A0	P0D0	E6	A8	P21	P23
PWM0L	P3A1	P0D1	E5	B8	P23	P24
PWM1H	P3A2	P0D2	C2	B6	P24	P25
PWM1L	P3A3	P0D3	C1	A5	P25	P26
PWM2H	P3A4	P0D4	D2	B5	P26	P27
PWM2L	P3A5	P0D5	D1	A4	P27	P28
PWMenb	P3A6	P0D6	E4	B4	P28	P30
PHT1Z	P3A7	P0D7	E3	A3	P30	P31
PHT1A	P3B0	P0D8	G2	C1	P31	P32
PHT1B	P3B1	P0D9	G1	C2	P32	P33
PHT0A	P3B2	P0D10	H5	D1	P33	P34
PHT0B	P3B3	P0D11	J6	D2	P35	P35
AMP0	P3B4	P0D12	H4	E1	P36	P39
AMP1	P3B5	P0D13	H3	E2	P40	P40
AMP2	P3B6	P0D14	H2	F2	P41	P41
AMP3	P3B7	P0D15	H1	G1	P44	P43
eeSK	P0C3	P1D0	C10	B9	P20	P22
eeCS	P0C2	P1D1	D10	A9	P18	P21
eeDI	P0C1	P1D2	F10	B10	P17	P20
eeDO	P0C0	P1D3	E10	B11	P15	P19
ssCK	P0B7	P1D4	E8	A11	P14	P17
ssCS	P0B6	P1D5	E7	A12	P13	P16
ssDI	P0B5	P1D6	A6	B13	P12	P15
ssDO	P0B4	P1D7	B6	A13	P11	P14
ppCLK	P3C0	P1D8	K4	G2	P46	P44
ppDIN	P3C1	P1D9	K3	K1	P47	P45
TxD	P3C2	P1D10	K2	K2	P50	P46
RxD	P3C3	P1D11	K1	L1	P51	P48
tstLED	P3C4	P1D12	L5	L2	P52	P49
stsLED	P3C5	P1D13	L4	M1	P53	P50
RxLED	P3C6	P1D14	L3	M2	P55	P51
TxLED	P3C7	P1D15	L2	N1	P56	P52

eM_DEMO 電路配置	eMotion 舊版定義	piDSP 新版定義	XC2V250 -FG456	XC3S1000 -FG456	XC3S200 -TQ144	XC95144XL -TQ144
xPA0	P2A0	P2D0	AA3	N2	P59	P53
xPA1	P2A1	P2D1	Y4	T1	P60	P54
xPA2	P2A2	P2D2	AA4	T2	P63	P56
xPA3	P2A3	P2D3	AB4	U2	P68	P57
xPC0	P2A4	P2D4	W5	V1	P69	P58
xPC1	P2A5	P2D5	Y5	V2	P70	P59
xPC2	P2A6	P2D6	AA5	W1	P73	P60
xPC3	P2A7	P2D7	AB5	W2	P74	P61
--	P2B0	P2D8	V6	Y1	P76	P64
--	P2B1	P2D9	V7	Y2	P77	P66
--	P2B2	P2D10	W6	AA3	P78	P68
mCS	P2B3	P2D11	Y6	AB4	P79	P69
mA15	P2B4	P2D12	V9	AA4	P80	P70
mA16	P2B5	P2D13	V10	AB5	P82	P71
mA17	P2B6	P2D14	W10	AA5	P83	P74
mA18	P2B7	P2D15	Y10	AA6	P84	P75
CS0	P2C0	ADC0	AA10	AB8	P85	P76
SDIN0	P2C1	ADC1	AB10	AA8	P86	P77
CS1	P2C2	ADC2	U10	AB9	P87	P78
SDIN1	P2C3	ADC3	U11	AA9	P89	P79
SCLK	P2C4	ADC4	V11	AB10	P90	P80
CS2	P2C5	ADC5	W11	AA10	P92	P81
SDOUT	P2C6	ADC6	Y11	AB11	P93	P82
LDAC#	P2C7	ADC7	AA11	AA11	P95	P83

eM_DEMO 電路配置	eMotion 舊版定義	piDSP 新版定義	XC2V250 -FG456	XC3S1000 -FG456	XC3S200 -TQ144	XC95144XL -TQ144
xPB0	P1A0	PxA0	M1	M3	P135	P85
xPB1	P1A1	PxA1	M2	M4	P132	P86
yPB2	P1A2	PxA2	M3	N3	P131	P87
yPB3	P1A3	PxA3	M4	N4	P130	P88
yPB1	P1A4	PxA4	M5	T4	P129	P91
yPB4	P1A5	PxA5	M6	T5	P128	P92
yPB0	P1A6	PxA6	N1	U3	P127	P93
yPB5	P1A7	PxA7	N2	U4	P125	P94
yPB6	P1B0	PxB0	U13	V3	P119	P95
yPB7	P1B1	PxB1	V13	V4	P118	P96
xPB2	P1B2	PxB2	W13	W3	P116	P97
xPB3	P1B3	PxB3	Y13	W4	P113	P98
xPB4	P1B4	PxB4	AA13	Y3	P112	P100
xPB5	P1B5	PxB5	AB13	Y4	P96	P101
yPC3	P1B6	PxB6	U12	Y5	P97	P102
yPC2	P1B7	PxB7	V12	W5	P98	P103
yPC1	P1C7	PxC0	AB17	W11	P108	P113
yPC0	P1C6	PxC1	AA17	Y11	P107	P112
yPA0	P1C5	PxC2	Y17	W10	P105	P111
yPA1	P1C4	PxC3	W17	Y10	P104	P110
yPA2	P1C3	PxC4	AB18	W9	P103	P107
yPA3	P1C2	PxC5	AA18	W8	P102	P106
xPB6	P1C1	PxC6	Y18	W6	P100	P105
xPB7	P1C0	PxC7	W18	Y6	P99	P104

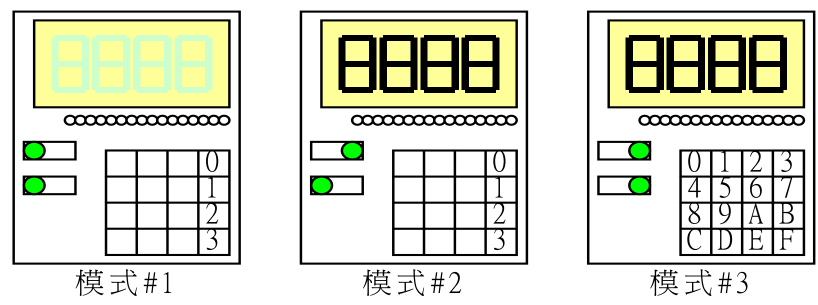
eM_DEMO模組

eM_DEMO模組的電路配置圖如下：



其中：

1.eM_DEMO模組的基本功能包括：
  A.將xPA/xPB/xPC等16個I/O訊號引出，控制LED和鍵盤電路。
  B.將AD/DA訊號引出，並放大成+/-10V的工作範圍。
2.LED和鍵盤可以選擇三種模式，由LED和KEY兩組開關作選擇，圖示如下：



3.模式#1：4鍵輸入並搭配16點的LED燈號顯示。其中：
  16點LED：分別對映到xPA[0~3]、xPB[0~7]和xPC[0~3]。
  4組按鍵：分別對映到xPC[0~3]。
4.模式#2：4鍵輸入並搭配四個7段式數字顯示。其中：
  數字顯示：由xPA[0~3]作數字掃瞄，而由xPB[0~7]作7段式LED控制。
  4組按鍵 ：分別對映到xPC[0~3]。
5.模式#3：16鍵輸入並搭配四個7段式數字顯示。其中：
  數字顯示：由xPA[0~3]作數字掃瞄，而由xPB[0~7]作7段式LED控制。
  16組按鍵：由xPA[0~3]作鍵盤掃瞄，而由xPC[0~3]作鍵盤讀取。
6.根據"VHDL與數位邏輯設計"的章節，其中：
  第1章至10章的實驗採用模式#1，而
  第11章實驗採用模式#2，而
  第12章之後的實驗都採用模式#3。